소프트웨어공학

텀프로젝트 보고서

소프트웨어공학 01분반

소프트웨어학부

20202475 이동훈

목차

1. 개요

본 프로젝트는 2022학년도 1학기 소프트웨어공학 텀프로젝트를 설명하는 문서이며, 프로젝트 전 과정에 해당하는 요구 정의 및 분석 산출물, 설계 산출물, 구현 산출물 및 구현 결과를 설명하는 내용을 담고 있다.

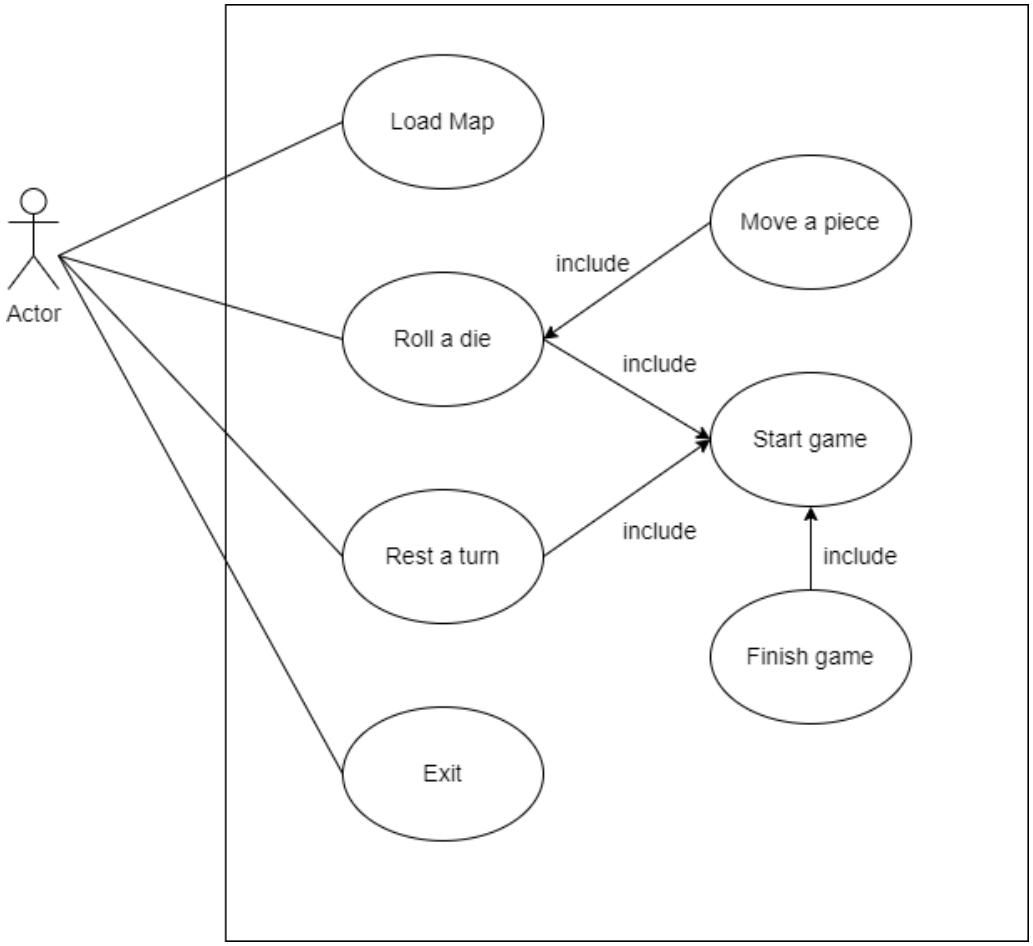
2장에서는 프로젝트에서 필요한 기능들에 대한 요구 정의 및 해당 요구들에 대한 분석에 관련된 자료들에 대해 설명하고 있다. 해당 프로그램에 대한 사용 행동을 설명하고 있는 유스케이스 다이어그램과 명세, 프로그램에 사용될 모델에 대한 기본적인 분석이 담겨 있는 도메인 모델, 프로그램이 어떻게 실행되는지에 대한 간단한 시스템 시퀀스 다이어그램, 각 유스케이스에서 필요한 메소드들에 대한 간단한 정의인 Operation Contract 에 대해 설명하고 있다.

3장에서는 2장에서 분석한 내용들을 기반으로 프로젝트의 구조를 설계한 자료들에 대해 설명하고 있다. Class들의 구조에 대해 설계한 Design Class Diagram, 프로그램 실행 흐름에 대해 설계한 Sequence Diagram, 프로그램의 각 단계에 맞는 상태를 나타낸 State Chart 에 대해 설명하고 있다.

마지막 4장에서는 구현된 프로그램에 대해 설명하고, 실행되는 과정과 흐름, 게임 작동 방법, 테스트 결과에 대해 보여주고 있다.

2. 요구 정의 및 분석 산출물

1) Use Case Diagram



Actor가 직접적으로 해야 할 행동은 위의 그림처럼 4가지로 정의할 수 있다. Load Map의 경우, 게임을 본격적으로 시작하기 전, 게임 보드판을 임의로 바꿀 수 있다. 사용자가 맵을 선택하면 프로그램에서 해당 파일을 기반으로 한 보드판을 만들게 된다. 게임을 시작한 이후에는 크게 두 가지 행동을 할 수 있는데, 주사위를 던져 말을 움직이거나, 한 턴을 쉴 수 있다. Roll a die를 하게 되면, 주사위를 던져 나온 수 만큼 말을 움직이게 된다.(Move a piece) Rest a turn을 하게 되면 다음 차례로 넘어가게 된다. 게임에 있는 모든 말이 종료 지점에 들어오게 되면 게임은 알아서 종료가 된다. 프로그램을 종료하고 싶다면 종료 버튼을 눌러 프로그램을 종료할 수 있다.

2) Use Case Description

아래 표들은 위의 Diagram에서 나온 Use Case 7가지에 대한 상세한 설명을 나타내고 있다.

UC1	Load Map
Scope	board game application
Level	user-goal level
Actor	user
Stakeholders	user : want to set the map of playing board games
Precondition	start game' is not processing
Postcondition	selected map is changed by chosen map
Main Scenario	
1)	click the load button
2)	choose the map file in the computer
3)	computer will read the file and make into map file
4)	save the map file in the game

Load Map Use Case에 대한 설명이다. 게임이 시작되기 전에 맵을 선택할 수 있으며, Load 버튼을 클릭하게 되면 파일 선택 창이 뜨게 되고, 해당 창에서 파일을 선택할 수 있다. 이후 게임을 시작하게 되면 선택한 파일을 기반으로 한 맵이 생성된다.

UC2	Start Game
Scope	board game application
Level	user-goal level
Actor	user
Stakeholders	user : want to start the game
Precondition	execute the program
Postcondition	make a new window to play game
Main Scenario	
1)	click the start button
2)	computer will make a game with loaded map
3)	set the waiting mode to set how many players will join the game
4)	click the game start button to play the game

Start Game Use Case에 대한 설명이다. 시작 버튼을 누르면 저장되어 있는 맵을 불러와 게임 시작 전 화면이 뜨게 된다.

UC3	Roll a die
Scope	board game application
Level	user-goal level
Actor	user
Stakeholders	user : want to move own piece
Precondition	start game' is done
Postcondition	ready to execute 'move a piece'
Main Scenario	
1)	click the roll button
2)	can get a random number that he can move
3)	ready to move a piece

Roll a Die Use Case에 대한 설명이다. 게임 시작 후 roll 버튼을 누르게 되면 무작위의 숫자가 뜨게 된다.

UC4	Move a piece
Scope	board game application
Level	user-goal level
Actor	user
Stakeholders	user : want to move own piece
Precondition	roll a die' is done
Postcondition	ready to next turn
Main Scenario	
1)	input the sequence of direction alphabet
2)	if it can move normally, move a piece

Move a piece Use Case 에 대한 설명이다. Roll 버튼 클릭 이후에 제공된 숫자 만큼의 이동 횟수가 주어지게 되고, 횟수에 맞는 이동 방향 문자열을 입력하면 해당 문자열이 인식하는 대로 말이 움직이게 된다.

UC5	Rest a turn
Scope	board game application
Level	user-goal level
Actor	user
Stakeholders	user : want to remove a bridge card
Precondition	start game' is done and 'roll a die' or 'move a piece' is not processing
Postcondition	ready to next turn
Main Scenario	
1)	click the rest button
2)	remove one bridge card

Rest a turn Use Case에 대한 설명이다. Roll 버튼 대신 Rest 버튼을 누르면 차례가 넘어가게 되고 해당 플레이어의 다리 카드를 한 장 삭제해준다.

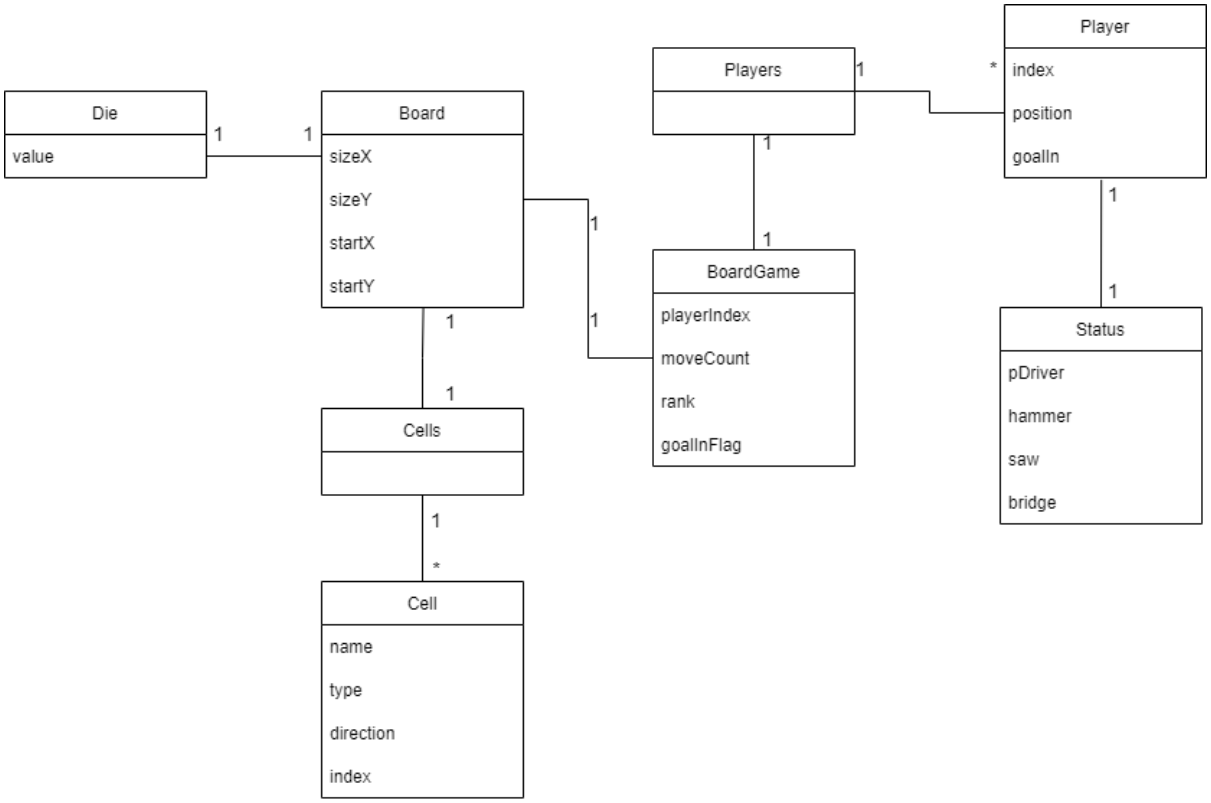
UC6	Finish game
Scope	board game application
Level	subfuction
Actor	system
Stakeholders	system : want to show the result
Precondition	start game' is done and 'roll a die' cannot start
Postcondition	reset the game and go to the first page
Main Scenario	
1)	show the result in new page
2)	click the first button to go back

Finish Game Use Case에 대한 설명이다. 모든 말이 도착 지점에 들어가면 게임이 알아서 종료하게 되고 결과 페이지를 보여주게 된다.

UC7	Exit game
Scope	board game application
Level	user-level goal
Actor	user
Stakeholders	user : want to exit the game
Precondition	nothing is processing
Postcondition	exit the program
Main Scenario	
1)	click the exit button
2)	exit the program

Exit Game Use Case에 대한 설명이다. 종료 버튼을 누르면 프로그램이 종료하게 된다.

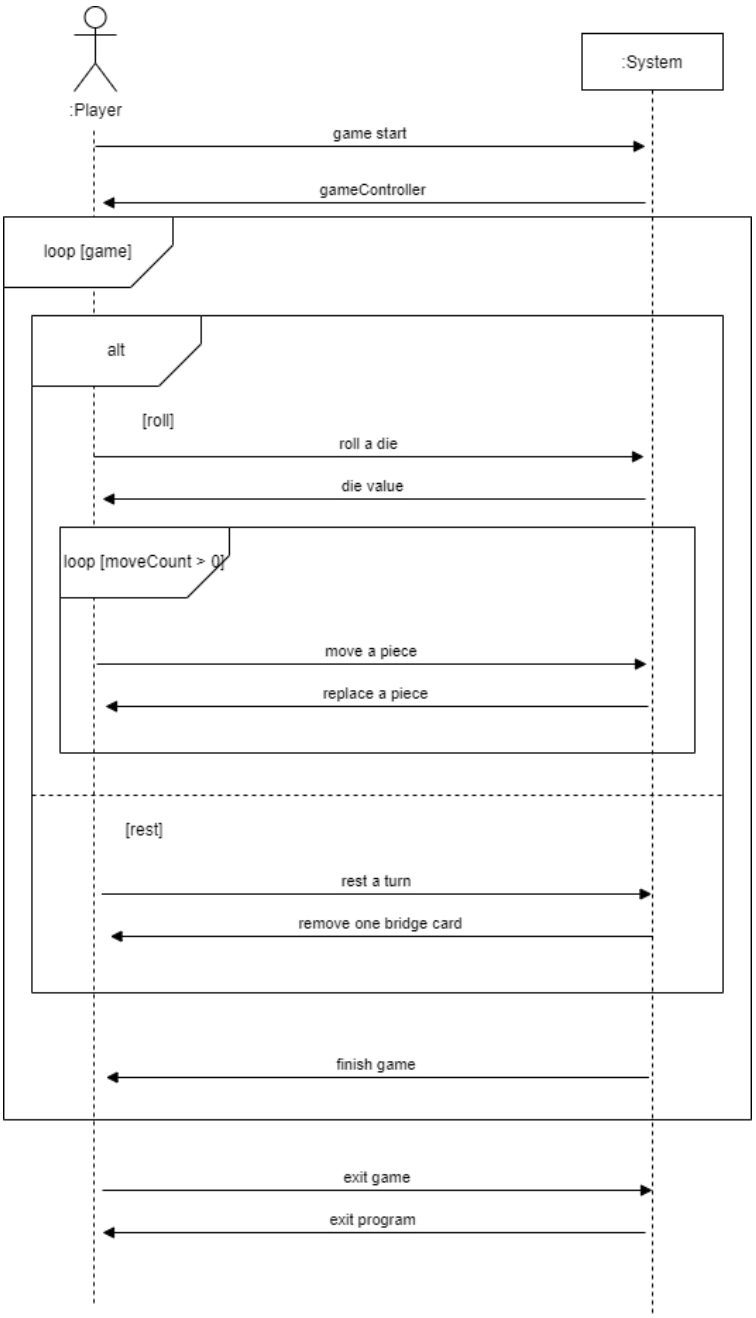
3) Domain Model



위의 Use Case 분석을 토대로 필요한 Model은 다음과 같다. 게임을 진행할 BoardGame Model, 보드게임을 할 수 있는 판인 Board Model, 판을 구성하는 Cells Model과 한 칸을 구성하는 Cell Model, 주사위를 의미하는 Die Model, 게임에 참가한 사람들인 Players Model, 사용자 한 명을 구성하는 Player Model, 해당 사용자의 현재 상태를 의미하는 Status Model이 있다.

각 모델들에 필요한 데이터는 위의 그림과 같다. BoardGame의 경우 현재 차례의 플레이어의 번호, 현재 차례에 이동할 수 있는 이동 횟수, 현재 도착한 사람의 명수, 한명이라도 골인한 사람이 있는지 확인하는 변수가 있다. Board의 경우 해당 맵의 사이즈와, 시작 지점의 위치를 저장하고 있다. Cell의 경우 해당 칸의 종류, 진행 가능한 방향, 칸의 번호 등을 저장하고 있다. Die의 경우 주사위를 굴렸을 때 나오는 값을 저장하고 있다. Player의 경우 본인의 번호, 본인이 위치한 칸의 번호, 도착했는지 여부를 저장하고 있다. 마지막으로 Status의 경우, 각 도구의 획득 횟수 및 다리 카드의 개수를 저장하고 있다.

4) System Sequence Diagram



Game start를 하게 되면 system에서는 게임 전체를 통솔하는 game controller를 넘겨주게 된다. 사용자가 UI에서 제공하는 모든 활동들은 controller에게 전달이 되고, controller에서 게임을 진행하게 된다. 게임이 끝날 때 까지 사용자는 해당 차례의 플레이어가 어떤 행동을 할 지 선택하게 되는데, roll을 선택하게 되면 주사위를 던지고 해당 칸수 만큼 움직이게 되고, rest를 선택하게 되면 한 턴 넘겨주면서 다리 카드를 하나 삭제한다. 게임이 끝나게 되면 System에서 먼저 알려주게 되고, 이후 게임을 종료할 수 있다.

5) Operation Contract

Operation	loadMap(mapFile: File)
Cross Reference	Use Cases: Load Map
Preconditions	game is not starting
Postconditions	map file is created

loadMap의 경우 맵 파일을 선택해서 들고가면 해당 파일을 토대로 맵 데이터를 제작하게 된다.

Operation	gameStart(mapFile: File)
Cross Reference	Use Cases: Start Game
Preconditions	load map is not processing
Postconditions	Cells is created by mapFile
	Board created cells which is made by mapFile
	BoardGame created Board made
	boardgame created players whose size is that scan the player number

gameStart의 경우 제작된 맵 데이터를 통해 게임판을 구성하게 되고, 이외의 기초 설정들과 함께 게임을 시작할 준비를 하게 된다.

Operation	roll()
Cross Reference	Use Cases: Roll a die
Preconditions	game is processing and not rest
Postconditions	get a random number
	set a moveCount which we get previous step

Roll의 경우 버튼을 누르면 1-6의 무작위 숫자를 얻게 되고 해당 숫자를 통해 움직일 수 있는 값을 설정하게 된다.

Operation	move(moveCount: int)
Cross Reference	Use Cases: Move a piece
Preconditions	roll is done and already get a die value
	calculate the moveCount which is considered with bridge card count
Postconditions	check the sequence of direction that can available to move
	move a piece to destination cell
	set the player next

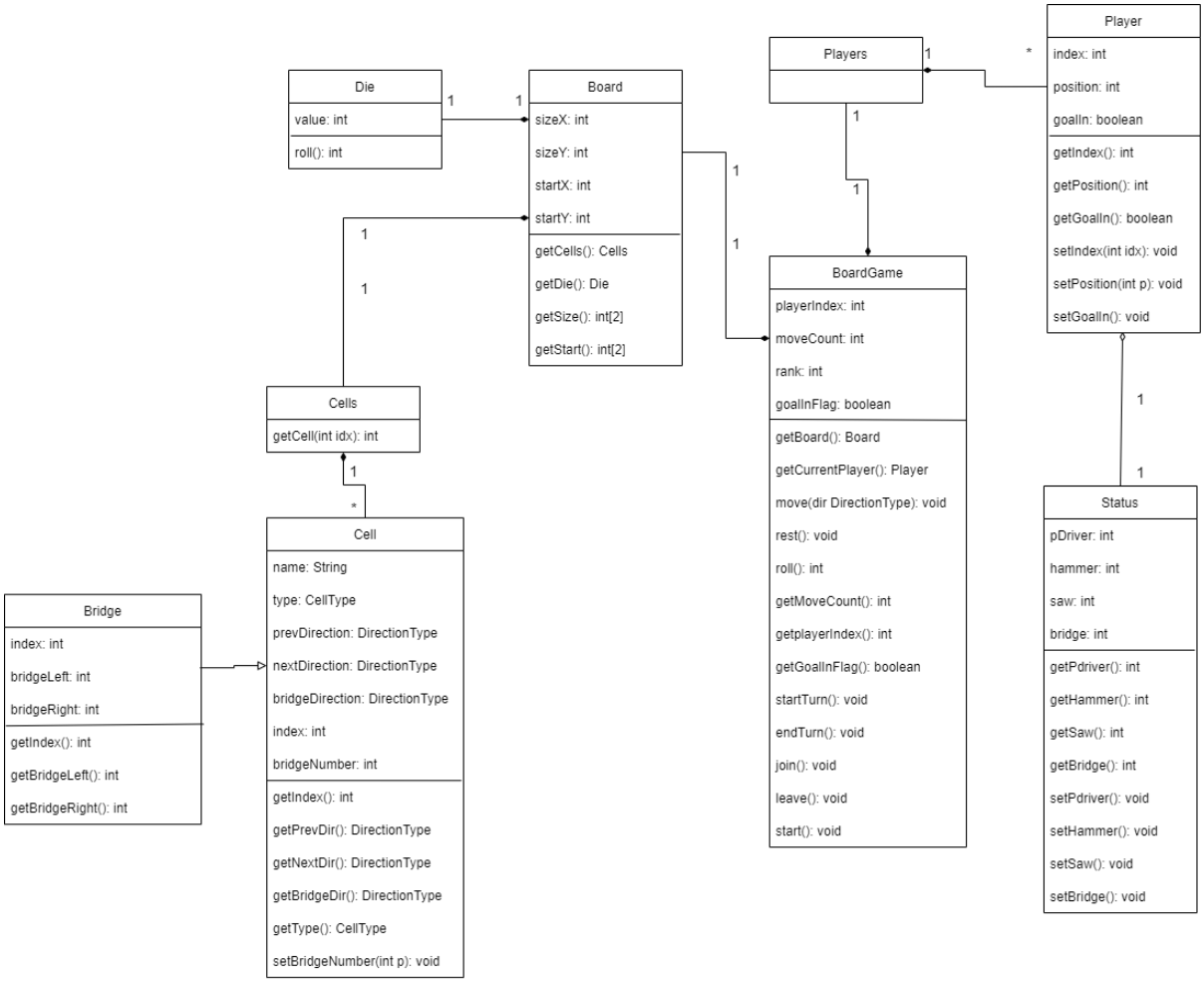
Move의 경우 위에서 얻은 무작위의 숫자를 파라미터로 받아 말을 움직일 수 있는 횟수를 설정하고 해당 횟수 만큼의 이동을 하게 된다.

Operation	rest()
Cross Reference	Use Cases: Rest a turn
Preconditions	roll and move is done or just start game
Postconditions	remove a currentPlayer's bridge card
	set the player next

Rest의 경우 한 턴을 쉬고 다리 카드를 하나 없애게 된다.

3. 설계 산출물

1) Design Class Diagram



Design Class의 경우, 위에서 만든 Domain Model을 토대로 각 Class에서 사용하는 method와 Class 사이의 관계 및 개수 등을 포함하여 만든 Diagram이다.

BoardGame의 경우 해당 게임에서 사용하고 있는 board를 얻는 getBoard 메소드, 현재 플레이어를 얻는 getCurrentPlayer 메소드, 기본 동작에 해당하는 move, roll, rest 메소드, 이동 횟수를 얻는 getMoveCount 메소드, 턴을 시작하고 종료하는 startTurn, endTurn 메소드, 게임에 참여하고 나가는 join, leave 메소드, 게임을 시작하는 start 메소드 등이 있다.

Board의 경우, 해당 보드를 구성하고 있는 Cells를 얻는 getCells 메소드, 해당 보드에 있는 주사위를 얻는 getDie 메소드, 해당 맵의 사이즈와 시작 지점을 알 수 있는 getSize, getStart 메소드가 있다.

Die에는 굴릴 수 있는 roll 메소드가 있고, Cells에는 특정 index에 해당하는 셀을 얻을 수 있는 getCell 메소드가 있다. 해당 메소드가 필요한 이유는 이후 코드 설명에 하도록 한다.

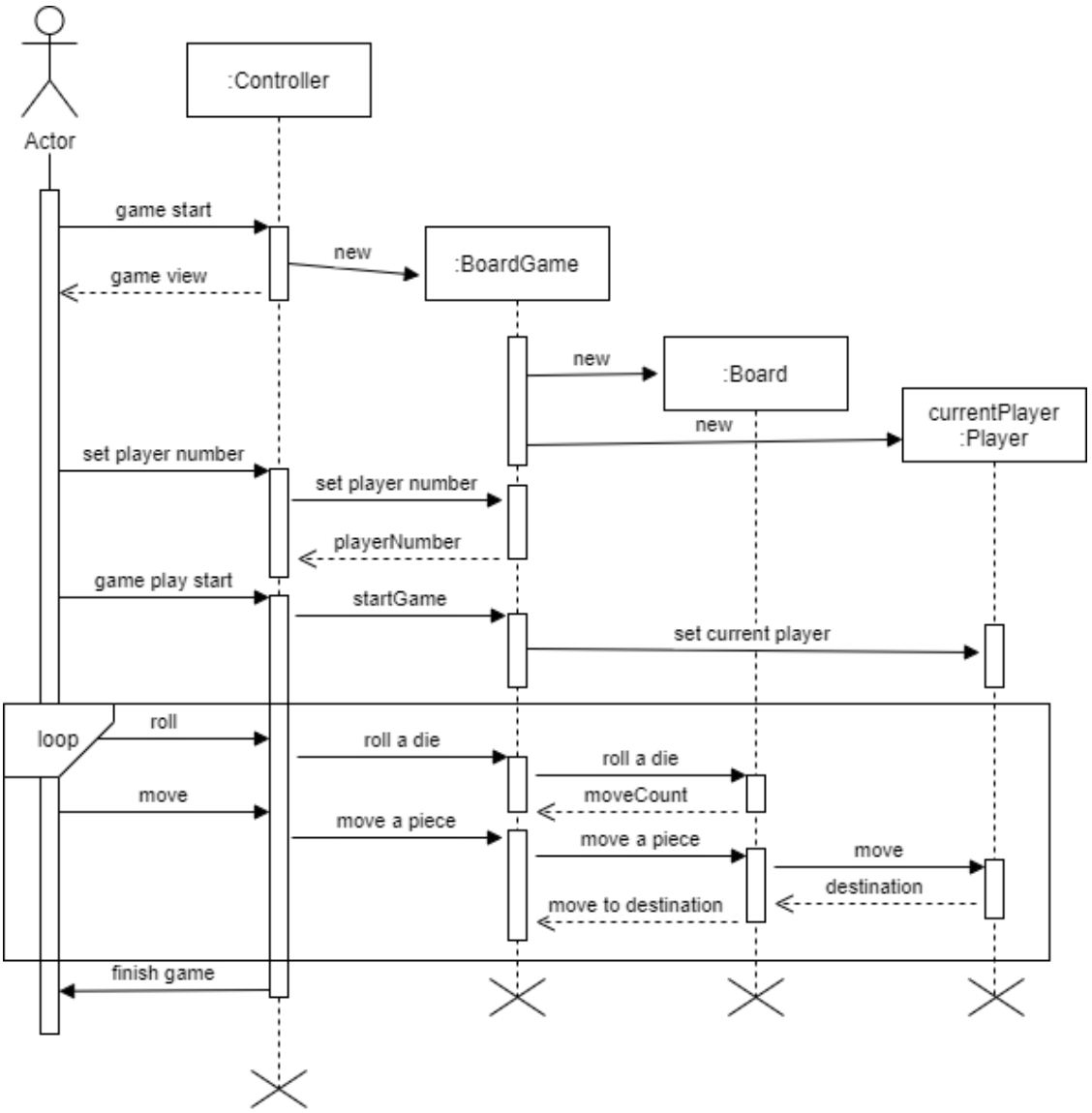
Cell Class에서는 해당 Cell의 번호를 얻는 getIndex, 해당 Cell에서 진행가능한 방향을 얻는 getPrevDir, getNextDir, getBridgeDir, Cell의 종류를 알 수 있는 getType, 해당 Cell과 이어진 다리를 연결시켜주는 setBridgeNumber 메소드가 있다.

Bridge Class는 Cell class에서 정의한 메소드 이외에도 해당 다리에 양 옆의 Cell을 얻을 수 있는 getBridgeLeft, getBridgeRight 메소드가 있다.

Player Class에서는 해당 플레이어의 번호, 위치, 골인 여부를 확인할 수 있는 getIndex, getPosition, getGoalIn 메소드가 있고, 해당 값들을 설정할 수 있는 setIndex, setPosition, setGoalIn 메소드가 있다.

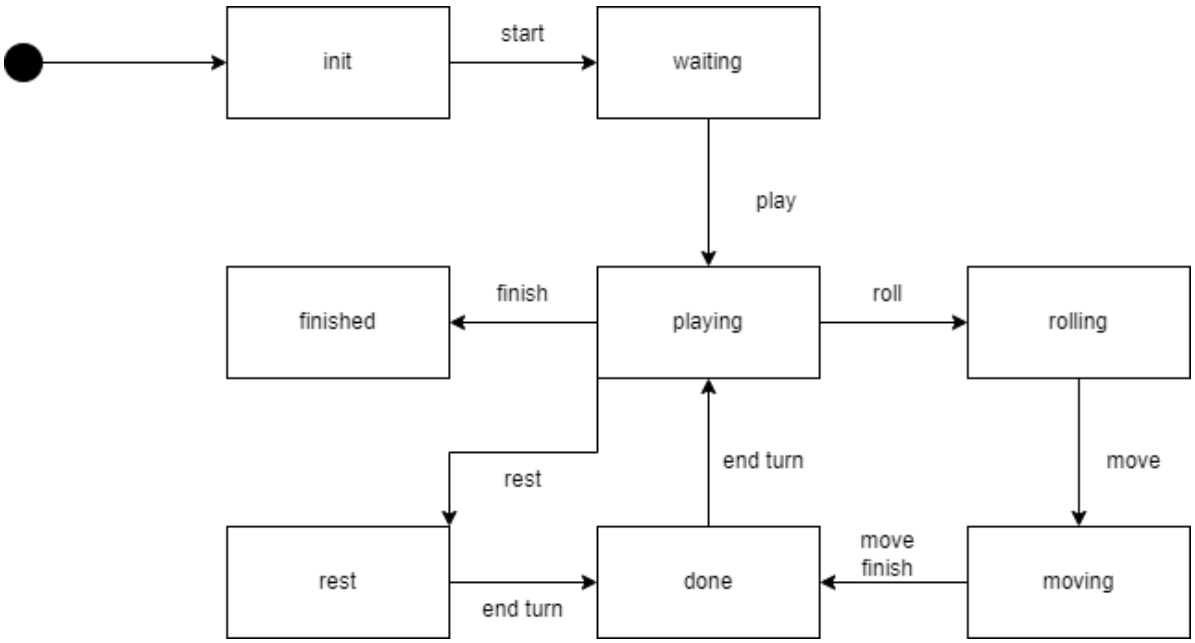
Status Class는 각 카드의 개수를 설정할 수 있는 getPdriver, getHammer, getSaw, getBridge, setPdriver, setHammer, setSaw, setBridge 메소드가 존재한다.

2) Sequence Diagram



다음으로 설계한 자료는 Sequence Diagram이다. 위의 자료는 System Sequence Diagram을 기반으로 System을 더 세분화하여 해당 Class들에서 어떤 동작이 일어나고 있는지를 더욱 자세히 나타낸 자료이다. Loop가 시작하기 전까지는 게임을 시작해서 Class들을 만들고, 게임에 참여할 플레이어의 수를 정하고, 게임을 시작하기 까지의 동작이 담겨 있고, loop 내부에는 게임을 시작해서 주사위를 던져서 말을 움직이는 동작들이 들어 있다.

3) State Chart



위의 그림은 프로젝트 프로그램의 State를 Chart로 나타낸 자료이다. 프로그램을 시작하게 되면 초기 상태인 init 상태가 된다. 이후 게임을 시작하면 게임을 본격적으로 시작하기 전 세팅 상태인 waiting 상태가 된다. 이후 게임을 시작하게 되면 playing 상태가 되고, 주사위를 던지면 rolling, 말을 움직이기 시작하면 moving, 움직임이 끝나면 done 상태가 되고, 게임이 끝나지 않았다면 다시 playing 상태가 된다. Roll 대신 rest를 선택하면 rest 상태 이후 차례가 끝났기 때문에 done 상태가 되고 다시 같은 구조를 통해 playing 상태가 된다. 게임이 종료되면 finished 상태가 되고 게임이 종료된다.

4. 구현 산출물

1) 프로그램 소스 코드

(1) Model

A. BoardGame.java

<table>
<tr>
<td>

```
public class BoardGame implements Cloneable{
    private Board board;
    private ArrayList<Player> players;
    private int playerIndex;
    private Player currentPlayer;
    private int moveCount;
    private int rank;
    private boolean goalInFlag;
```

</td>
<td>board : 게임을 진행할 보드판
players : 게임에 참여할 player 배열
playerIndex : 현재 차례인 player의 index
currentPlayer : 현재 차례인 player
moveCount : 현재 player가 움직일 수 있는 횟수
rank : 다음으로 들어오는 player의 등수
- 현재까지 들어온 사람의 수
goalInFlag : 도착한 사람이 있는지에 대한 Flag</td>
</tr>
<tr>
<td>

```
public BoardGame() {
    board = new Board();
    players = new ArrayList<Player>();

    this.join(); this.join();
}
```

</td>
<td>BoardGame constructor
Board와 players에 새 객체 생성
최소 인원이 2명이기 때문에 join 2번 실행</td>
</tr>
<tr>
<td>

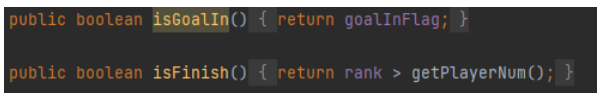
```
@Override
public BoardGame clone() throws CloneNotSupportedException {
    BoardGame clone = (BoardGame) super.clone();
    ArrayList<Player> tmp = new ArrayList<>();
    for (Player player : players) {
        tmp.add(player.clone());
    }
    clone.players = tmp;
    clone.currentPlayer = clone.players.get(playerIndex);
    clone.board = board.clone();
    return (BoardGame) clone;
}
```

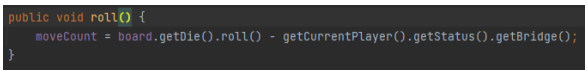
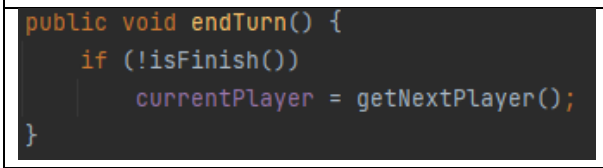
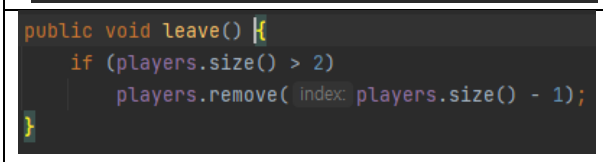
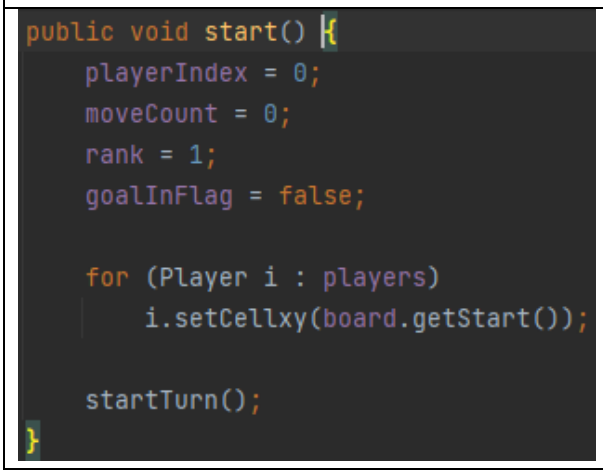
</td>
<td>말 이동시에 유효한 이동인지 확인하기 위해 복제된 객체를 통해 이동 유효성 검사를 하게 되는데 그때 사용되는 clone 메소드 - 안에 있는 멤버 변수 및 객체들도 복사를 진행하게 된다.</td>
</tr>
<tr>
<td>

```
public Board getBoard() { return board; }

public ArrayList<Player> getPlayers() { return players; }

public Player getCurrentPlayer() { return currentPlayer; }

public int[] getCurrentCell() { return currentPlayer.getCellxy(); }

public int getMoveCount() { return moveCount; }
```

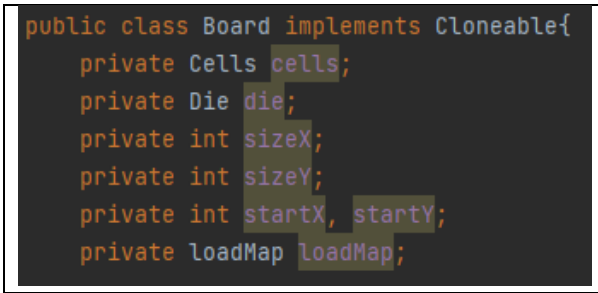
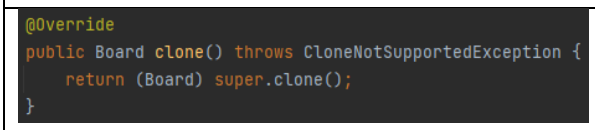
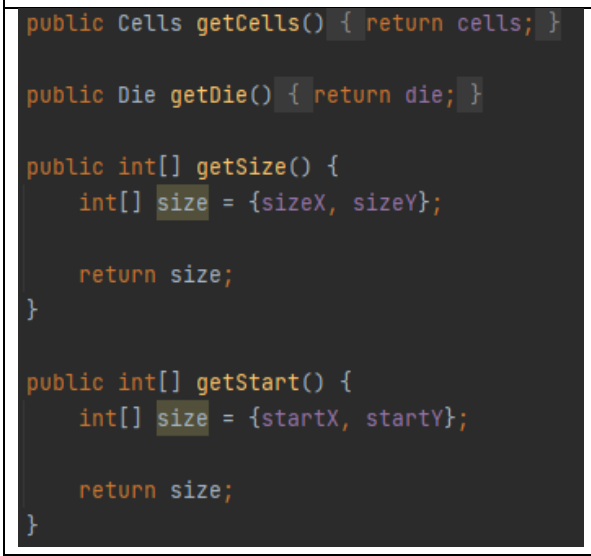
```
public int getPlayerNum() { return players.size(); }

public int getPlayerIndex() { return playerIndex; }

public int getRank() { return rank; }

public int[] getSize() { return board.getSize(); }
```

</td>
<td>옆의 메소드들은 모두 멤버 클래스 및 변수들을 반환하는 메소드이다.</td>
</tr>
</table>

	getStatus 메소드는 현재 참가한 플레이어들의 카드 개수에 대한 정보를 반환하는 함수이다. Status의 경우 player에게 포함되어 있기 때문에 players 배열을 통해 하나하나 참조하여 가지고 온 후 새 객체를 만들어 반환하게 된다.
	getNextPlayer는 턴이 끝나고 다음 차례의 player를 얻는 메소드이다. 이미 골인한 플레이어는 건너뛰고, playerIndex가 참여한 플레이어보다 클 경우 다시 첫번째로 초기화도 시킨다.
	setMoveCount의 경우 파라미터 count 값으로 설정하게 된다. setGoalInFlag 메소드를 호출하면 한명이라도 골인을 했다는 것으로 간주하고 Flag를 true로 변경한다. addRank 메소드는 도착할때마다 호출해 도착한 사람 수를 확인할 수 있다. addBridge 메소드는 다리를 건넌 플레이어에게 다리 카드를 한 장 추가해주는 기능을 한다. resetBridge의 경우 다리를 건넌 여부를 확인하기 위한 flag를 초기화한다. setCrossBridge 함수는 해당 플레이어가 다리를 왼쪽으로 건너고 있는지, 오른쪽으로 건너고 있는지 확인하기 위해 설정한 함수이다.
	isGoalIn는 누군가 도착했는지 여부를 판단하는 메소드, isFinish는 모두가 도착했는지 여부를 판단하는 메소드이다.
	Move 메소드는 이동할 cell의 index와 이동할 방향을 입력받아 해당 플레이어의 위치와 셀 번호를 설정한다.

	Roll 메소드는 주사위를 굴려서 나온 숫자와 현재 플레이어가 가진 다리 카드의 개수를 통해 이동가능한 숫자를 계산한다.
	Rest 메소드는 현재 플레이어의 다리 카드를 하나 삭제한다. startTurn 메소드의 경우 현재 플레이어를 설정해준다.
	endTurn 메소드는 게임이 끝나지 않았다면 다음 플레이어를 현재 플레이어로 지정해준다.
	Join 메소드는 최대 인원 4명이 넘지 않는 선에서 플레이어를 한 명 추가한다.
	Leave 메소드는 최소 인원 2명이 부족하지 않는 선에서 플레이어를 한 명 삭제한다.
	Start 메소드는 게임을 시작하기에 앞서 각종 변수들을 초기화 하고 플레이어들의 말을 시작점에 위치해준다.

B. Board Model

	Cells : 맵을 구성하고 있는 cell의 배열 Die : 주사위 객체 sizeX, sizeY : 맵 사이즈 startX, startY : 시작점 좌표 loadMap : 맵을 불러오기 위한 객체
	Board constructor 맵을 불러오고, cells에 불러온 맵에 대한 정보를 저장 새로운 주사위 객체도 저장 사이즈와 시작점에 대한 정보 저장
	위에서 설명한 이유와 동일하게 복제를 위한 clone 메소드 정의
	getCells : Cells 객체 반환 getDie : 주사위 객체 반환 getSize : 맵 사이즈 반환 getStart : 시작점 좌표 반환

C. Cell

<table>
<tr>
<td>

```
public class Cell {
    private CellType type;
    private DirectionType previous;
    private DirectionType next;
    private DirectionType bridge;
    private int bridgeNumber;
    private int bridgeLeft;
    private int bridgeRight;

    private int index;
```

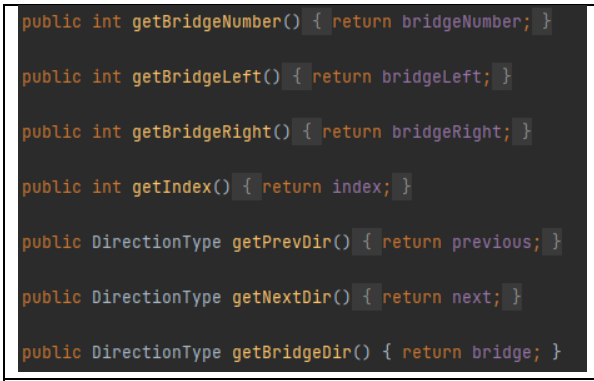
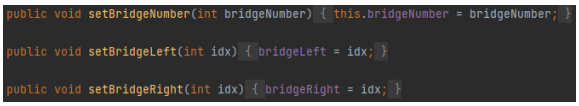
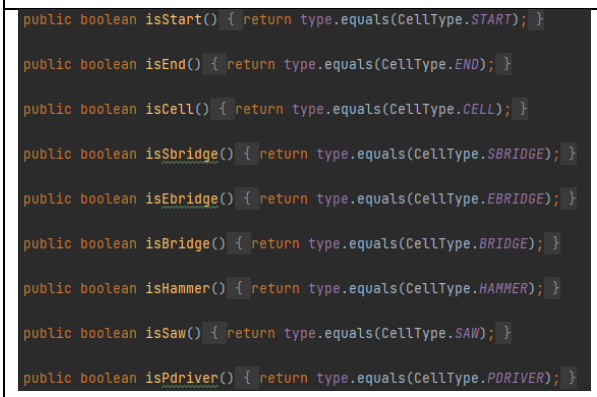
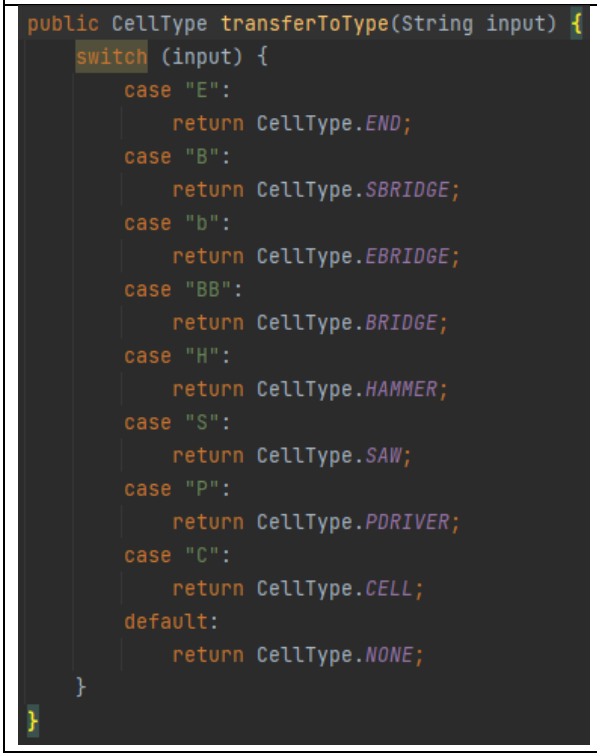
</td>
<td>CellType : 해당 셀의 종류
Previous : 이전 셀로의 진행방향
Next : 다음 셀로의 진행방향
Bridge : 다리 셀로의 진행방향
bridgeNumber : 연결된 다리 셀의 번호
bridgeLeft : 다리 셀일 때 왼쪽 셀의 번호
bridgeRight : 다리 셀일 때 오른쪽 셀의 번호 (Bridge class를 분리하려 했으나, 배열에 같이 저장하기 위해 분리하지 않음)</td>
</tr>
<tr>
<td>

```
public Cell(ArrayList<String> input, int i) {
    index = i;
    type = transferToType(input.get(0));
    switch (type) {
        case SBRIDGE -> {
            previous = transferToDir(input.get(1));
            next = transferToDir(input.get(2));
            bridge = DirectionType.RIGHT;
        }
        case EBRIDGE -> {
            previous = transferToDir(input.get(1));
            next = transferToDir(input.get(2));
            bridge = DirectionType.LEFT;
        }
        default -> {
            previous = transferToDir(input.get(1));
            next = transferToDir(input.get(2));
            bridge = DirectionType.NONE;
        }
    }
}
```

</td>
<td>Cell constructor
셀에 대한 정보를 가지고 있는 string 배열과 해당 셀의 인덱스를 파라미터로 받아옴
Index를 저장하고, 셀 타입을 설정한다.
첫번째 string이 셀의 타입에 대한 정보를 저장하고 있으므로 해당 데이터에 맞게 정보를 가공하여 저장한다.</td>
</tr>
<tr>
<td>

```
public Cell(ArrayList<String> input, CellType type, int i) {
    index = i;
    switch (type) {
        case START -> {
            this.type = CellType.START;
            previous = DirectionType.NONE;
            next = transferToDir(input.get(1));
            bridge = DirectionType.NONE;
        }
        case END -> {
            this.type = CellType.END;
            previous = transferToDir(input.get(1));
            next = DirectionType.NONE;
            bridge = DirectionType.NONE;
        }
    }
}
```

</td>
<td>Cell constructor
Start의 경우, saw와 타입 입력이 같기 때문에 길이로 판단을 해야 하는데, 이 경우 호출을 다르게 해서 구분을 하고 있다.
End의 경우 이전 방향을 모르기 때문에 호출 시에 가공을 통해 입력을 시켜서 설정하게 된다.</td>
</tr>
</table>

	Get 메소드들을 통해 클래스에 있는 변수들의 값을 받아올 수 있다.
	setBridgeNumber의 경우 B나 b 셀에 연결할 다리 셀을 설정하고 있다. SetBridge 메소드는 다리 셀일 경우 양옆의 셀을 설정하는 메소드이다.
	Is 메소드들은 해당 셀이 어떤 타입인지 확인할 수 있는 메소드이다. 타입을 반환하지 않고 각 타입에 맞는 Boolean 메소드들을 지정해 상황에 맞게 호출해 사용할 수 있다.
	transferToType 메소드는 스트링 타입의 데이터를 CellType의 데이터로 변환하는 메소드이다. BB 타입의 입력은 새로 정의한 것인데 다리 셀은 입력이 되지 않으므로 지정하기 위해 설정했다.

```java
public DirectionType transferToDir(String input) {
    switch(input) {
        case "L":
            return DirectionType.LEFT;
        case "R":
            return DirectionType.RIGHT;
        case "U":
            return DirectionType.UP;
        case "D":
            return DirectionType.DOWN;
        default:
            return DirectionType.NONE;
    }
}
```

transferToDir 메소드는 스트링 타입의 데이터를 DirectionType 데이터로 변환하는 메소드이다.

D. Cells

```java
public class Cells extends ArrayList<Cell>{

    public Cell getCell(int idx) {
        for (Cell i : this) {
            if (i.getIndex() == idx)
                return i;
        }
        return null;
    }
}
```

Cells class는 ArrayList<Cell> 형태를 재정의한 클래스이다. 재정의한 이유는 다리 셀의 인덱스를 음수로 저장하고 있는데, 음수 인덱스를 참조할 수 없기 때문에 ArrayList 안의 인덱스를 사용하지 않고 임의의 변수를 만들어서 사용하고 있다.

E. CellType

```java
public enum CellType {
    NONE, START, CELL, SBRIDGE, EBRIDGE, BRIDGE, HAMMER, SAW, PDRIVER, END
}
```

Cell의 종류를 지정하기 위해 enum으로 데이터를 만들어서 저장하고 있다.

F. DirectionType

```java
public enum DirectionType {
    LEFT, RIGHT, UP, DOWN, NONE
}
```

방향의 종류를 지정하기 위해 enum으로 데이터를 만들어서 저장하고 있다.

G. Die

```java
public class Die {

    private Random random;
    private int value;

    public Die() { random = new Random(); }

    public int roll() {
        value = random.nextInt( bound: 6) + 1;
        return value;
    }
}
```

Random : 무작위 숫자를 생성하기 위한 Random 객체
Value : 주사위 값을 저장하기 위한 변수
Constructor : Random 객체 생성
Roll : value에 무작위 수를 저장하고 해당 수를 반환한다.

H. Player

```java
public class Player implements Cloneable{
    private Status status;
    private int index;
    private int[] cellxy;
    private int position;
    private boolean goalIn;
    private int bridgeFlag;
```

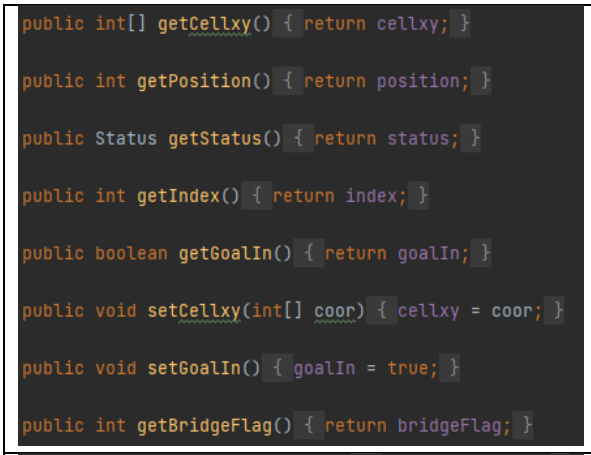
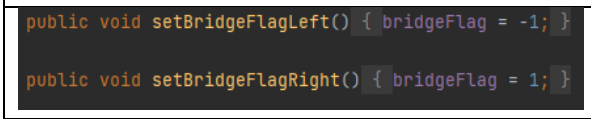
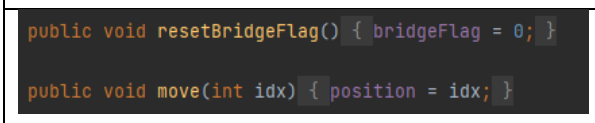
Status : 플레이어의 카드 상태
Index : 플레이어 번호
Cellxy : 플레이어의 위치
Position : 플레이어가 위치한 셀 번호
goalIn : 플레이어의 도착 여부
bridgeFlag : 플레이어가 어느 다리 위에 있는지에 대한 정보

```java
public Player(int idx) {
    status = new Status();
    position = 0;
    goalIn = false;
    bridgeFlag = 0;
    index = idx;
}
```

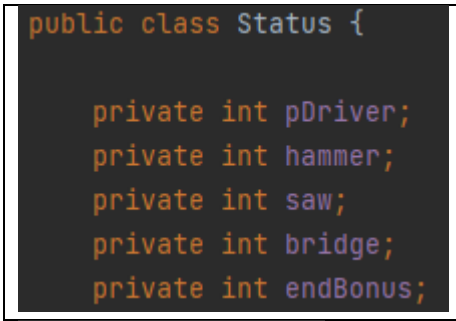
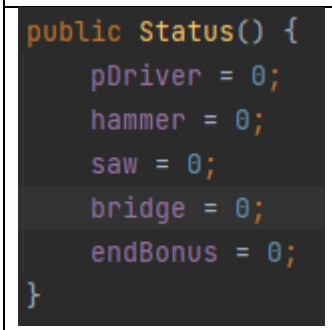
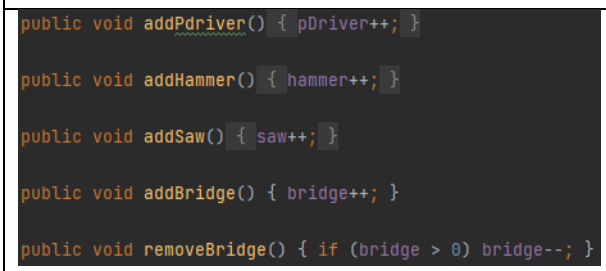
Constructor
Status 는 새 객체 생성
Position은 첫번째 셀로 설정
goalIn flag false 설정
bridgeFlag 초기화
index 설정

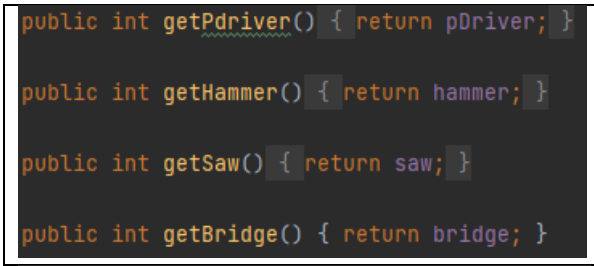
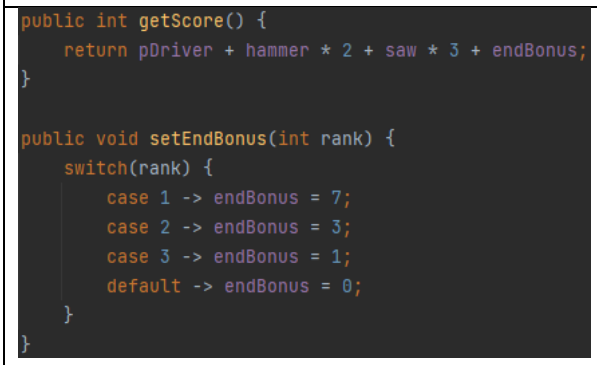
```java
@Override
public Player clone() throws CloneNotSupportedException {
    return (Player) super.clone();
}
```

위에서 말한 복제를 위한 clone 메소드

	Get 메소드들을 통해 클래스 멤버 변수들에 대한 정보를 얻을 수 있음 Set 메소드들을 통해 클래스 멤버 변수 값 변경 가능
	다리를 건너고 있는지를 확인하기 위한 플래그 설정
	resetBridgeFlag를 통해 다리를 건너거나 건너지 않은 후 초기화 진행 move 메소드를 통해 해당 idx 셀로 이동

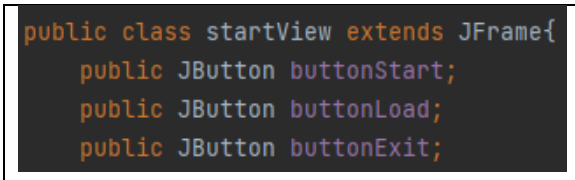
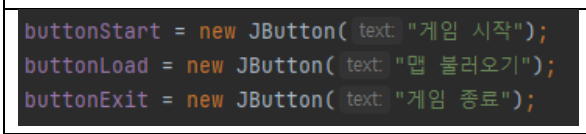
I. Status

	endBonus : 골인한 등수에 따른 도착 점수 나머지 변수 : 카드 개수
	Constructor 모든 카드 및 endBonus 0으로 초기화
	Add 메소드들을 통해 카드 한장 추가 removeBridge는 한 턴 쉰 후 다리 카드 한장 제거

	Get 메소드들을 통해 해당 카드의 개수 얻을 수 있음
	Getscore 메소드를 통해 지금까지의 총점을 얻을 수 있음 setEndBonus는 등수를 파라미터로 입력받아 도착 보너스를 저장하고 있음

(2) View

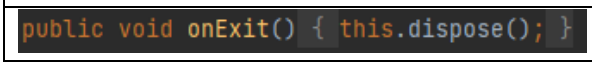
A. startView

	Buttonstart : 시작 버튼 Buttonload : 맵 불러오기 버튼 Buttonexit : 종료 버튼
	Pn1 : 프로그램 이름과 이름을 넣을 패널 Pn2 : 버튼을 넣을 패널 titlePanel : 프로그램 이름을 넣을 패널 namePanel : 이름을 넣을 패널
	버튼에 jbutton 객체 생성
	제목 설정을 위한 스트링 생성 Jlabel에 넣을 스트링 지정 및 객체 생성 라벨 폰트 및 가운데 정렬 설정

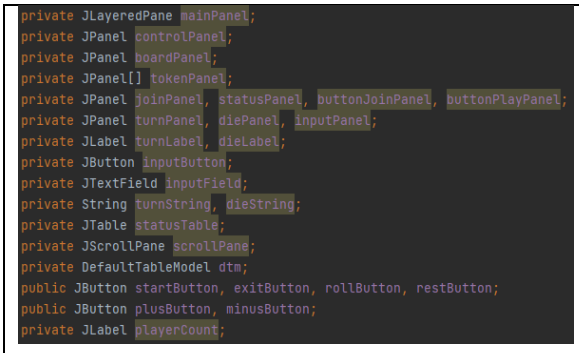
`pn1.setLayout(new BoxLayout(pn1, BoxLayout.Y_AXIS));` `titlePanel.add(lb);` `pn1.add(titlePanel);` `pn1.add(new JPanel());` `namePanel.add(name);` `pn1.add(namePanel);` `pn1.add(pn2);`	Pn1 레이아웃을 박스 레이아웃(세로축) 지정 Titlepanel에 lb 추가 Pn1에 titlepanel 추가 Pn1에 pn2 추가
`BorderLayout fl = new BorderLayout( hgap: 0, vgap: 120);` `pn2.setBorder(BorderFactory.createEmptyBorder( top: 30, left: 0, bottom: 30, right: 0));` `pn2.add(buttonStart);` `pn2.add(buttonLoad);` `pn2.add(buttonExit);`	프레임에 지정할 borderlayout 생성 Pn2에 패딩 지정 및 버튼 추가
`this.setLayout(fl);` `this.add(new JPanel(), BorderLayout.NORTH);` `this.add(pn1, BorderLayout.CENTER);` `this.add(new JPanel(), BorderLayout.SOUTH);` `this.setSize( width: 1200, height: 900);` `this.setVisible(true);` `this.setDefaultCloseOperation(EXIT_ON_CLOSE);`	프레임에 레이아웃 생성 및 패널 추가 프레임 보이기 설정 및 창 닫기 설정
`public void onExit() { this.dispose(); }`	종료를 위한 onExit 메소드 정의

B. resultView

`public class resultView extends JFrame{` `  private JPanel panel;` `  private ArrayList<Player> list;` `  public JButton button;`	Panel : frame에 넣을 Jpanel List : 플레이어 배열 Button : 처음으로 돌아가기 위한 버튼
`class MyComparator implements Comparator<Player> {` `  @Override` `  public int compare(Player o1, Player o2) {` `  if (o1.getStatus().getScore() > o2.getStatus().getScore())` `  return -1;` `  else if (o1.getStatus().getScore() > o2.getStatus().getScore())` `  return 1;` `  else return 0;` `  }` `}`	플레이어 배열을 점수 순으로 정렬하기 위한 comparator 정의
`public resultView(BoardGame game) {` `  panel = new JPanel();` `  panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));` `  list = game.getPlayers();` `  list.sort(new MyComparator());`	Panel 에 패널 객체 생성 Boxlayout(세로축)으로 패널 레이아웃 지정 List에 game에 참여한 player 배열 불러오기 List를 점수 순으로 정렬
`for (Player i : list) {` `  JLabel label = new JLabel();` `  label.setText("Player " + i.getIndex() + " Score : " + i.getStatus().getScore());` `  label.setHorizontalAlignment(JLabel.CENTER);` `  panel.add(label);` `}`	반복문을 통해 한명씩 panel에 추가

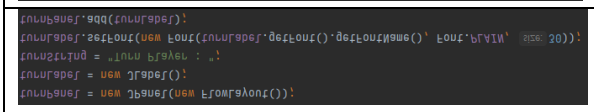
	Button 객체 생성 및 패널에 추가 프레임에 패널 추가 프레임 사이즈 및 보이기, 창 닫기 설정
	프로그램 종료를 위한 onExit 메소드

C. gameView

	mainPanel : 보드게임 판과 플레이어 말을 포함하는 패널 controlPanel : 오른쪽에 나타나는 현재 플레이어, 주사위 값, 움직이기 입력칸, 현재 게임 진행상황, 주사위 던지기 쉬기 버튼을 포함하는 패널 boardPanel : 보드게임 판 패널 tokenPanel : 플레이어 말 패널 joinPanel : 게임 시작 전 플레이어 수를 지정하기 위한 정보를 위한 패널 statusPanel : 게임 진행상황 패널 buttonJoinPanel : 게임 시작 또는 종료 버튼 패널 buttonPlayPanel : 주사위 던지기 쉬기 버튼 패널 turnPanel : 현재 누구의 차례인지 에 대한 정보를 담는 패널 diePanel : 주사위 값에 대한 정보를 담은 패널 inputPanel : 움직이기 입력칸과 입력 버튼을 담은 패널 turnLabel, dielabel : 현재 차례, 주사위 값을 나타내는 라벨 inputbutton : 움직임 입력 후 설정을 위한

	버튼 inputfield : 움직임 입력을 위한 textfield turnstring, diestring : turnlabel, dielabel에 설정할 스트링 statusTable : 게임 진행상황 정보를 담는 table scrollpane : statusTable 설정을 위한 scrollpane dtm : statusTable 설정을 위한 테이블 모델 startButton : 시작버튼 exitButton : 종료버튼 rollButton : 주사위 굴리기 버튼 restButton : 한 턴 쉬기 버튼 plusButton : 한명 추가 버튼 minusButton : 한명 제거 버튼 playerCount : 현재 추가된 플레이어 수
`private boardGameController controller;` `private BoardGame game;` `private String[] header;`	Controller : 게임 진행을 위한 controller Game : 현재 진행중인 게임 Header : table 설정을 위한 스트링 배열
`public gameView() {` `  this.setLayout(new BorderLayout());` `  controller = new boardGameController();` `  game = controller.reset();` `  //Main Panel setting start` `  mainPanel = new JLayeredPane();` `  //Board Panel setting start` `  boardPanel = new JPanel();` `  boardPanel.setLayout(new GridBagLayout());`	프레임 레이아웃 설정 Controller 에 새로운 객체 생성 Game 에 게임 초기화 설정 패널들 객체 생성 및 레이아웃 설정
`GridBagConstraints gbc = new GridBagConstraints();` `gbc.fill = GridBagConstraints.BOTH;` `gbc.weightx = gbc.weighty = 1;` `gbc.gridwidth = gbc.gridheight = 1;` `gbc.ipadx = gbc.ipady = 0;`	GridBaglayout 설정을 위한 기본 설정
`Cells cells = game.getBoard().getCells();` `int[] start = game.getBoard().getStart();`	맵 ui를 띄우기 위한 기본 정보 불러오기

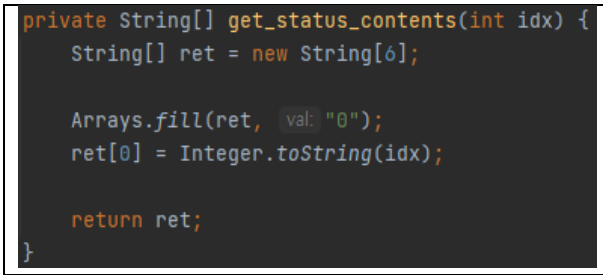
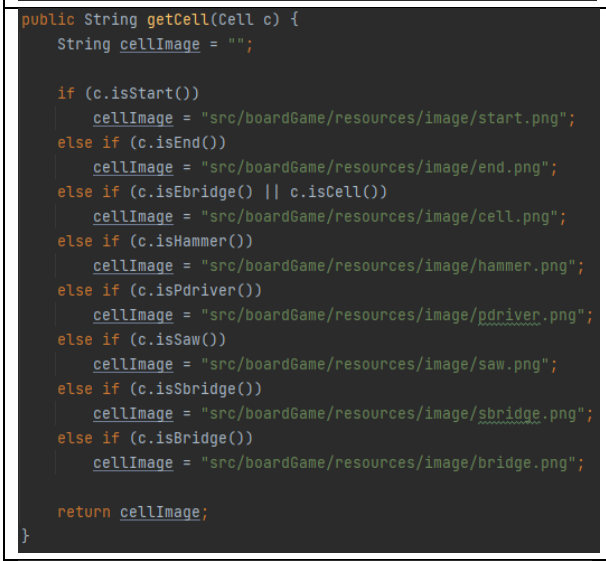
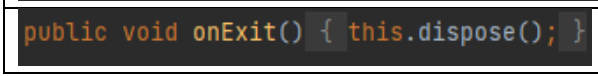
`` for (Cell i : cells) { if (i.isStart()) { gbc.gridx = start[0]; gbc.gridy = start[1]; JLabel imgLabel = new JLabel(); String cellImage = getCell(i); ImageIcon icon = new ImageIcon(cellImage); imgLabel.setIcon(icon); boardPanel.add(imgLabel, gbc); } ```	반복문을 통해 맵 생성 시도 현재 셀이 시작 셀일 경우 시작 좌표를 설정해 이미지 파일을 불러와 해당 칸에 설정
`` else if (!i.isBridge()){ if (i.getPrevDir() == DirectionType.LEFT) gbc.gridx++; else if (i.getPrevDir() == DirectionType.RIGHT) gbc.gridx--; else if (i.getPrevDir() == DirectionType.UP) gbc.gridy++; else if (i.getPrevDir() == DirectionType.DOWN) gbc.gridy--; JLabel imgLabel = new JLabel(); String cellImage = getCell(i); ImageIcon icon = new ImageIcon(cellImage); imgLabel.setIcon(icon); boardPanel.add(imgLabel, gbc); if (i.isSbridge()) { gbc.gridx++; imgLabel = new JLabel(); cellImage = "src/boardGame/resources/image/bridge.png"; icon = new ImageIcon(cellImage); imgLabel.setIcon(icon); boardPanel.add(imgLabel, gbc); gbc.gridx--; } } ```	다리 셀이 아닐경우 갈 수 있는 방향을 확인한 후 해당 좌표 설정 이후 이미지를 불러와 해당 위치에 추가 만약 해당 셀이 B 셀일 경우, 해당 위치의 오른쪽에 다리 셀 추가
`` boardPanel.setBounds(x: 20, y: 20, width: game.getSize()[0] * 60, height: game.getSize()[1] * 60); boardPanel.setOpaque(true); ```	보드 패널의 총 크기 설정 및 불투명도 설정
`` tokenPanel = new JPanel[4]; for(int i = 0; i < 4; i++) tokenPanel[i] = new JPanel(); for (JPanel i : tokenPanel) i.setBackground(new Color(r: 255, g: 0, b: 0, a: 0)); ```	토큰 패널에 객체 생성 및 배경 투명 설정
`` for(int i = 0; i < 4; i++) { JLabel imgLabel = new JLabel(); String cellImage = "src/boardGame/resources/image/token" + i + ".png"; ImageIcon icon = new ImageIcon(cellImage); imgLabel.setIcon(icon); tokenPanel[i].add(imgLabel); tokenPanel[i].setBounds(x: 14 + start[0] * 60, y: 10 + start[1] * 60, width: 60, height: 60); tokenPanel[i].setOpaque(true); tokenPanel[i].setVisible(false); mainPanel.add(tokenPanel[i]); } ```	토큰 이미지를 불러온 후 시작 위치에 맞게 위치 설정 후 패널에 추가하기
`` mainPanel.setBounds(x: 10, y: 10, width: 800, height: 800); mainPanel.add(boardPanel); ```	메인 패널 크기 설정 및 추가

`` controlPanel = new JPanel(); controlPanel.setLayout(new BoxLayout(controlPanel, BoxLayout.Y_AXIS)); ```	컨트롤 패널 생성 및 레이아웃 지정
	턴 패널 생성 및 내용물 설정
`` diePanel = new JPanel(); dieLabel = new JLabel(); dieString = "Left To Move : "; dieLabel.setFont(new Font(turnLabel.getFont().getFontName(), Font.PLAIN, size: 30)); diePanel.add(dieLabel); ```	다이패널 생성 및 내용물 설정
`` inputPanel = new JPanel(new FlowLayout()); inputField = new JTextField(columns: 10); inputButton = new JButton(text: "입력"); inputField.setBounds(x: 0, y: 0, width: 200, height: 40); inputPanel.add(inputField); inputPanel.add(inputButton); ```	인풋패널 생성 및 내용물 설정
`` inputButton.addActionListener(e -> { String input = inputField.getText(); if (input.length() == game.getMoveCount()) { boolean flag = true; boardGameController tmpc = null; BoardGame tmp = null; try { tmpc = controller.clone(); } catch (CloneNotSupportedException ex) { throw new RuntimeException(ex); } ```	인풋버튼 리스너 설정 -움직임 유효성 검사 시작 길이가 같으면 1차 통과 이후 움직임 유효성 검사 위해 복사 객체 생성
`` String[] moveDir = input.split(regex: ""); int[] prev = game.getCurrentCell(); for (String i : moveDir) { tmp = tmpc.move(i); if (tmp.getCurrentCell() != prev) { dieLabel.setText(dieString + (tmp.getMoveCount())); tokenPanel[tmp.getPlayerIndex()].setBounds(x: 14 + tmp.getCurrentCell()[0] * 60, y: 10 + tmp.getCurrentCell()[1] * 60, width: 60, height: 60); prev = tmp.getCurrentCell(); } else if (tmp.getMoveCount() == 0) break; else if (!tmp.getCurrentPlayer().getGoalIn()){ flag = false; dieLabel.setText(dieString + (game.getMoveCount())); tokenPanel[game.getPlayerIndex()].setBounds(x: 14 + game.getCurrentCell()[0] * 60, y: 10 + game.getCurrentCell()[1] * 60, width: 60, height: 60); break; } } ```	반복문을 통해 한칸씩 움직임을 실행하는데, 움직여졌다는 것은 움직일 수 있는 방향으로 갔다는 것이고, 움직여지지 않았다는 것은 올바르지 않은 방향이라는 뜻임. 이후 한번의 움직임마다 체크해서 한번이라도 움직이지 못했다면 불가능하단 뜻이므로 모두 취소. 끝까지 모두 잘 움직였다면 해당 복제 객체를 원래 객체로 저장하게 된다.

`` if (flag) { controller = tmpc; game = tmp; inputField.setText(""); controlPanel.revalidate(); controlPanel.repaint(); moveDone(); rollButton.setEnabled(true); restButton.setEnabled(true); } ```	복제 객체를 원래 객체에 저장하는 모습 및 세팅 초기화 및 값 설정
`` joinPanel = new JPanel(); joinPanel.setLayout(new FlowLayout()); ```	조인 패널 생성 및 레이아웃 지정
`` plusButton = new JButton(text: "+"); minusButton = new JButton(text: "-"); playerCount = new JLabel(text: "2"); plusButton.addActionListener(e -> { game = controller.join(); playerCount.setText(Integer.toString(game.getPlayerNum())); if (game.getPlayerNum() > dtm.getRowCount()) { dtm.addRow(get_status_contents(game.getPlayerNum())); } }); minusButton.addActionListener(e -> { game = controller.leave(); playerCount.setText(Integer.toString(game.getPlayerNum())); if (game.getPlayerNum() < dtm.getRowCount()) { dtm.removeRow(game.getPlayerNum()); } }); ```	조인 패널에 들어가는 요소들 생성 및 기본 설정 버튼에 리스너 설정해서 게임에 참여하는 인원 조정
`` joinPanel.add(minusButton); joinPanel.add(playerCount); joinPanel.add(plusButton); ```	조인 패널에 요소 추가
`` statusPanel = new JPanel(); statusPanel.setLayout(new GridLayout()); header = new String[]{"Player", "Bridge", "P-Driver", "Hammer", "Saw", "Score"}; dtm = new DefaultTableModel(header, rowCount: 0); statusTable = new JTable(dtm); dtm.addRow(get_status_contents(idx: 1)); dtm.addRow(get_status_contents(idx: 2)); DefaultTableCellRenderer tscr = new DefaultTableCellRenderer(); tscr.setHorizontalAlignment(SwingConstants.CENTER); TableColumnModel tcm = statusTable.getColumnModel(); for (int i = 0; i < tcm.getColumnCount(); i++) tcm.getColumn(i).setCellRenderer(tscr); statusTable.setRowHeight(30); scrollPane = new JScrollPane(statusTable); statusPanel.add(scrollPane); ```	Status패널 설정 및 레이아웃 지정 패널 안에 들어가는 테이블 설정 및 데이터 저장

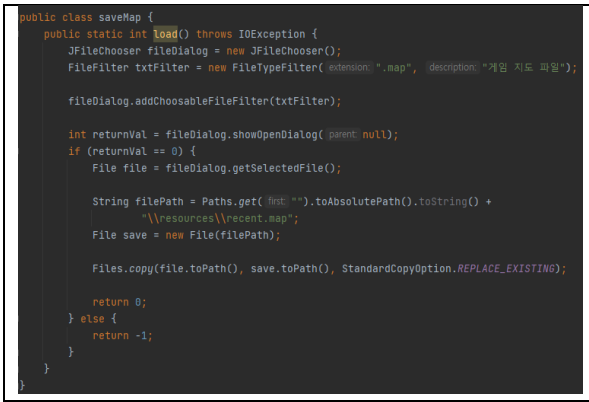
`` buttonJoinPanel = new JPanel(); buttonJoinPanel.setLayout(new FlowLayout()); startButton = new JButton(text: "START"); exitButton = new JButton(text: "EXIT"); startButton.addActionListener(e -> { change(); }); buttonJoinPanel.add(startButton); buttonJoinPanel.add(exitButton); ```	버튼조인패널 설정 및 레이아웃 지정,
`` buttonPlayPanel = new JPanel(); buttonPlayPanel.setLayout(new FlowLayout()); rollButton = new JButton(text: "ROLL"); restButton = new JButton(text: "REST"); rollButton.addActionListener(e -> { rollButton.setEnabled(false); restButton.setEnabled(false); game = controller.roll(); moving(); inputPanel.setVisible(true); }); restButton.addActionListener(e -> { game = controller.rest(); game = controller.endTurn(); inputPanel.setVisible(false); nextTurn(); }); buttonPlayPanel.add(rollButton); buttonPlayPanel.add(restButton); ```	버튼플레이패널 설정 및 내용 추가 안에 있는 버튼 두개에 리스너 설정 및 해당 행동에 대한 설정
`` controlPanel.add(joinPanel); controlPanel.add(buttonJoinPanel); ```	컨트롤 패널 요소 추가
`` this.add(new JPanel(), BorderLayout.NORTH); this.add(mainPanel, BorderLayout.CENTER); this.add(controlPanel, BorderLayout.EAST); this.setSize(width: 1200, height: 900); this.setVisible(true); this.setDefaultCloseOperation(EXIT_ON_CLOSE); ```	메인 프레임에 요소 추가 및 기본 설정

`` public void change() { controlPanel.removeAll(); controlPanel.add(turnPanel); controlPanel.add(diePanel); controlPanel.add(inputPanel); controlPanel.add(statusPanel); controlPanel.add(buttonPlayPanel); scrollPane.setPreferredSize(new Dimension(width: 350, height: 30)); game = controller.start(); turnLabel.setText(turnString + (game.getPlayerIndex() + 1)); dieLabel.setText(dieString); for (int i = 0; i < game.getPlayerNum(); i++) tokenPanel[i].setVisible(true); inputPanel.setVisible(false); controlPanel.revalidate(); controlPanel.repaint(); } ```	게임 스타트 버튼 눌렀을 때 패널 변환 및 기본 설정
`` public void setStatusTable(ArrayList<Status> list) { for (int i = 0; i < list.size(); i++) { dtm.setValueAt(list.get(i).getBridge(), i, column: 1); dtm.setValueAt(list.get(i).getPdriver(), i, column: 2); dtm.setValueAt(list.get(i).getHammer(), i, column: 3); dtm.setValueAt(list.get(i).getSaw(), i, column: 4); dtm.setValueAt(list.get(i).getScore(), i, column: 5); } } ```	게임 진행상황 테이블 값 설정 메소드
`` public void moving() { dieLabel.setText(dieString + (game.getMoveCount())); controlPanel.revalidate(); controlPanel.repaint(); } ```	말 움직이기 시작 전, roll 데이터 세팅
`` public void moveDone() { game = controller.moveAfter(); game = controller.endTurn(); inputPanel.setVisible(false); if (game.isFinish()) { controller.gameFinish(game); this.onExit(); } else nextTurn(); } ```	말 다 움직이고 난 후 후처리 및 게임 종료 여부 확인
`` public void nextTurn() { setStatusTable(game.getStatus()); game = controller.startTurn(); turnLabel.setText(turnString + (game.getPlayerIndex() + 1)); dieLabel.setText(dieString); controlPanel.revalidate(); controlPanel.repaint(); } ```	한 차례가 끝난 이후 다음 차례를 위한 세팅

	초기 게임 진행상황 테이블 설정
	초기 맵 세팅때 이미지 불러오기 위한 메소드
	프로그램 종료를 위한 onExit 메소드

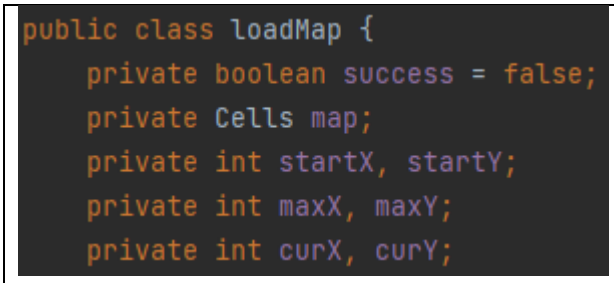
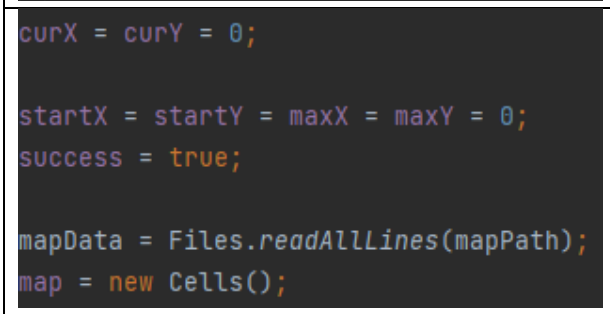
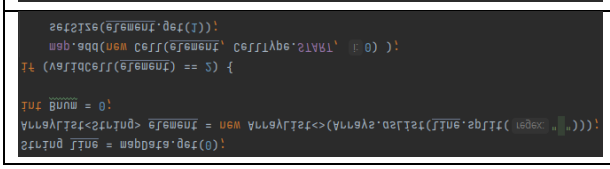
(3) Controller & System part

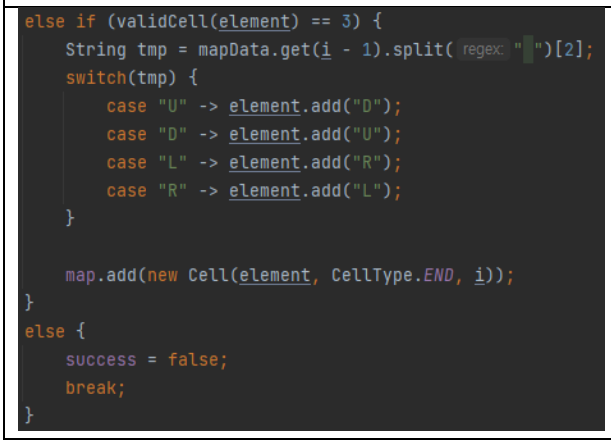
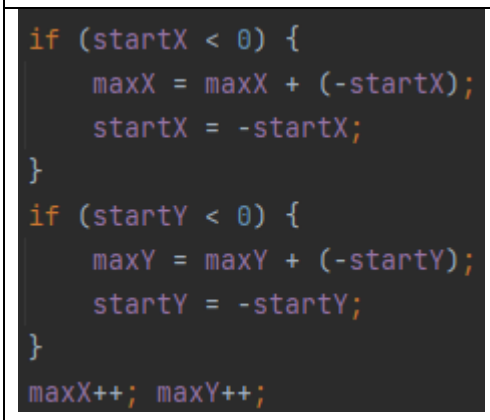
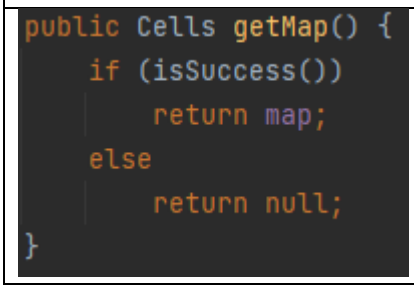
A. saveMap

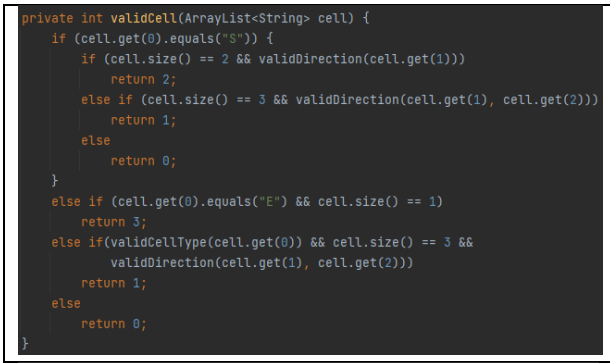
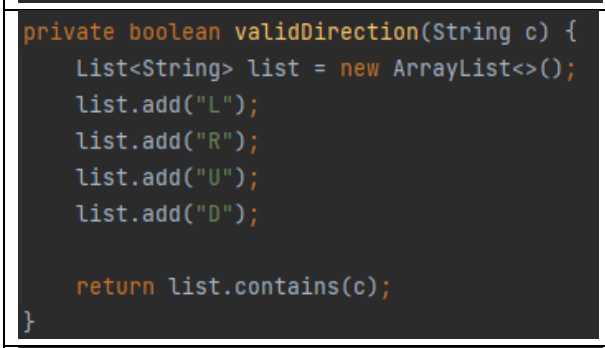
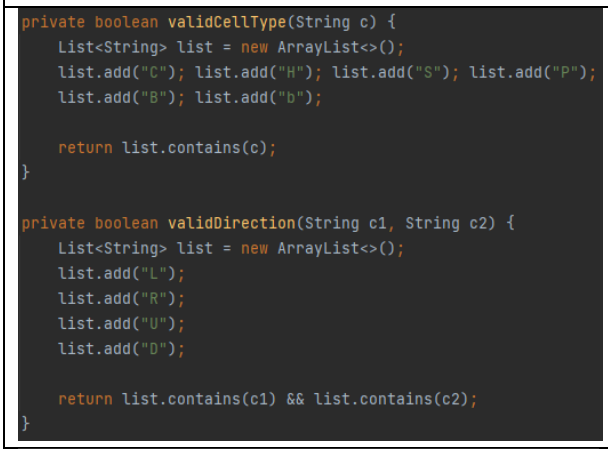
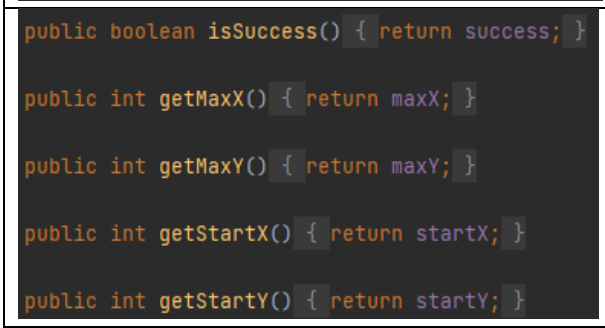
	새로운 맵 파일을 불러올 때, 기본적으로 게임은 recent.map을 불러오기 때문에 새 파일을 recent.map에 덮어쓰기 해야함. 그렇기 때문에 선택된 파일을 recent.map에 복사함.

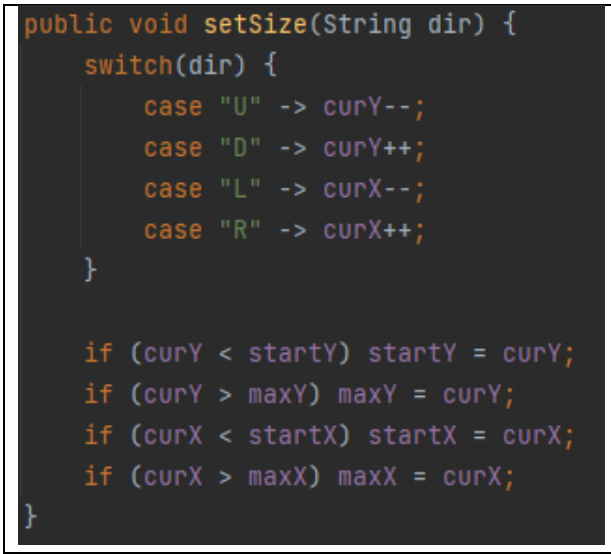
	fileFilter의 경우 .map 파일만을 검색하기 위해 설정한 클래스

B. loadmap

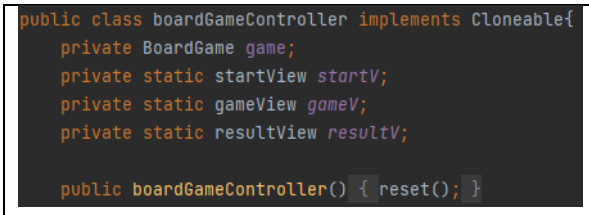
	맵을 프로그램 내에서 사용할 수 있는 데이터로 가공하기 위한 클래스 Success의 경우 가공 성공 여부 저장 startX, startY는 시작 좌표 maxX, maxY는 전체 맵 사이즈 curX, curY는 데이터를 하나씩 읽으면서 다닐 현재 좌표
	Recent.map 파일 불러오기 만약 없다면 default.map 파일을 불러옴
	읽기 전 기본 세팅 준비
	첫번째 입력된 셀이 시작 셀이라면 설정

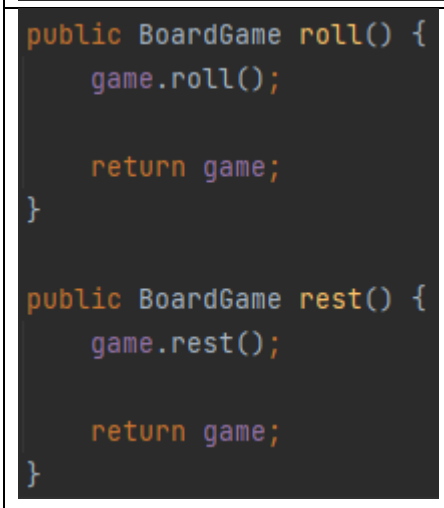
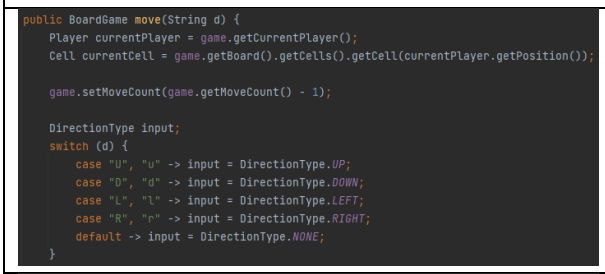
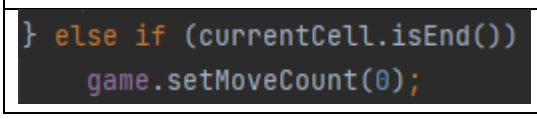
	한 줄 씩 입력 받아 해당되는 셀에 맞게 데이터 가공 후 map에 저장 validCell 값이 1인 경우 – 마지막 셀을 제외한 모든 셀 이후 B나 b 셀일 경우는 다리에 관한 정보 세팅 이후 해당 셀 맵에 저장
	validCell이 3일 경우 마지막 셀이므로 이전 방향을 토대로 마지막 셀의 이전 방향 등의 정보를 저장한 후 마지막으로 map에 저장 만약 그렇지 않다면 올바르지 않은 맵이므로 구성 실패 표시
	맵 사이즈와 시작 위치를 확인하기 위함 만약 처음으로 들어온 셀의 위치가 0,0 이 아닐 경우 늘어난 길이를 계산해 조정
	구성에 성공한 경우 맵을, 그렇지 않다면 null을 반환

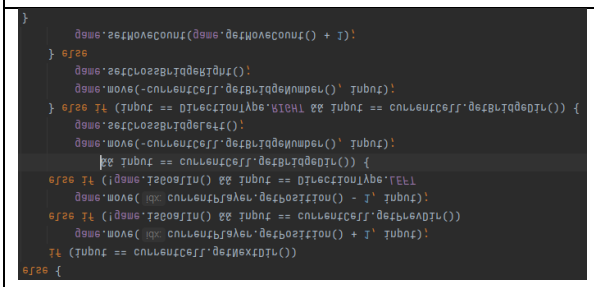
	해당 셀이 어떤 종류인지에 대한 값을 반환 시작셀 – 2 일반셀 – 1 도착셀 – 3 이외의 이상한 값 - 0
	유효한 방향이 들어왔는지 확인하는 메소드
	유효한 셀타입인지, 방향인지 확인하는 메소드
	해당 변수들에 대한 값을 받아올 수 있는 get 메소드

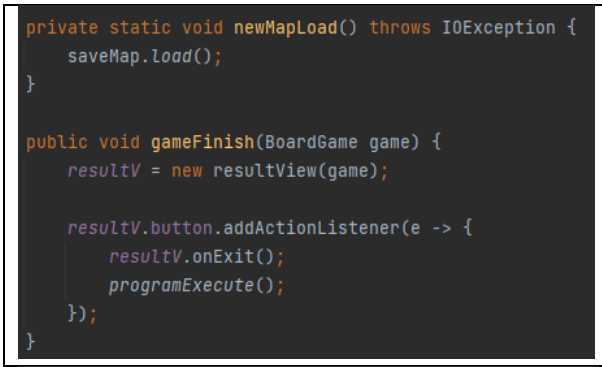
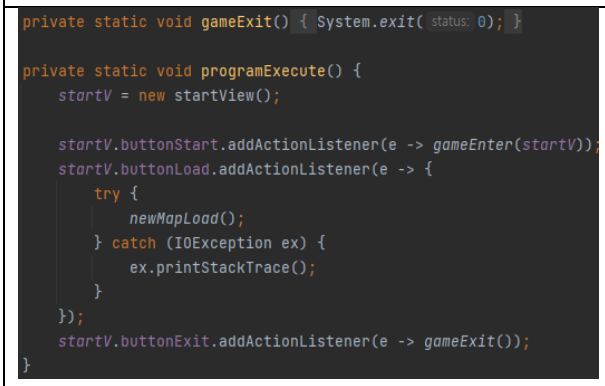
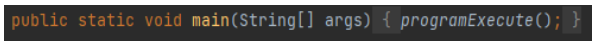
	맵 탐색을 위한 사이즈를 측정하기 위한 현재 위치 조정 메소드

C. boardGameController

	컨트롤러 클래스 정의 Game – 해당 게임 startV, gameV, result – ui에 필요한 뷰 constructor – reset 메소드 호출
	Reset – 새 보드게임 객체 생성 Start – game.start 호출

	Join – game.join 호출 Leave – game.leave 호출
	Roll – game.roll 호출 Rest – game.rest 호출
	현재 플레이어와 그 플레이어가 있는 위치를 받아온 후 해당 방향으로 움직임을 시도한다. 해당 위치를 받아옴
	현재 셀이 다리 셀인 경우, 양 옆을 확인하고 가능한 경우 움직임을, 가능하지 않다면 다시 횟수를 늘려서 움직이지 않음.
	만약 현재 셀이 마지막 셀이라면 움직임을

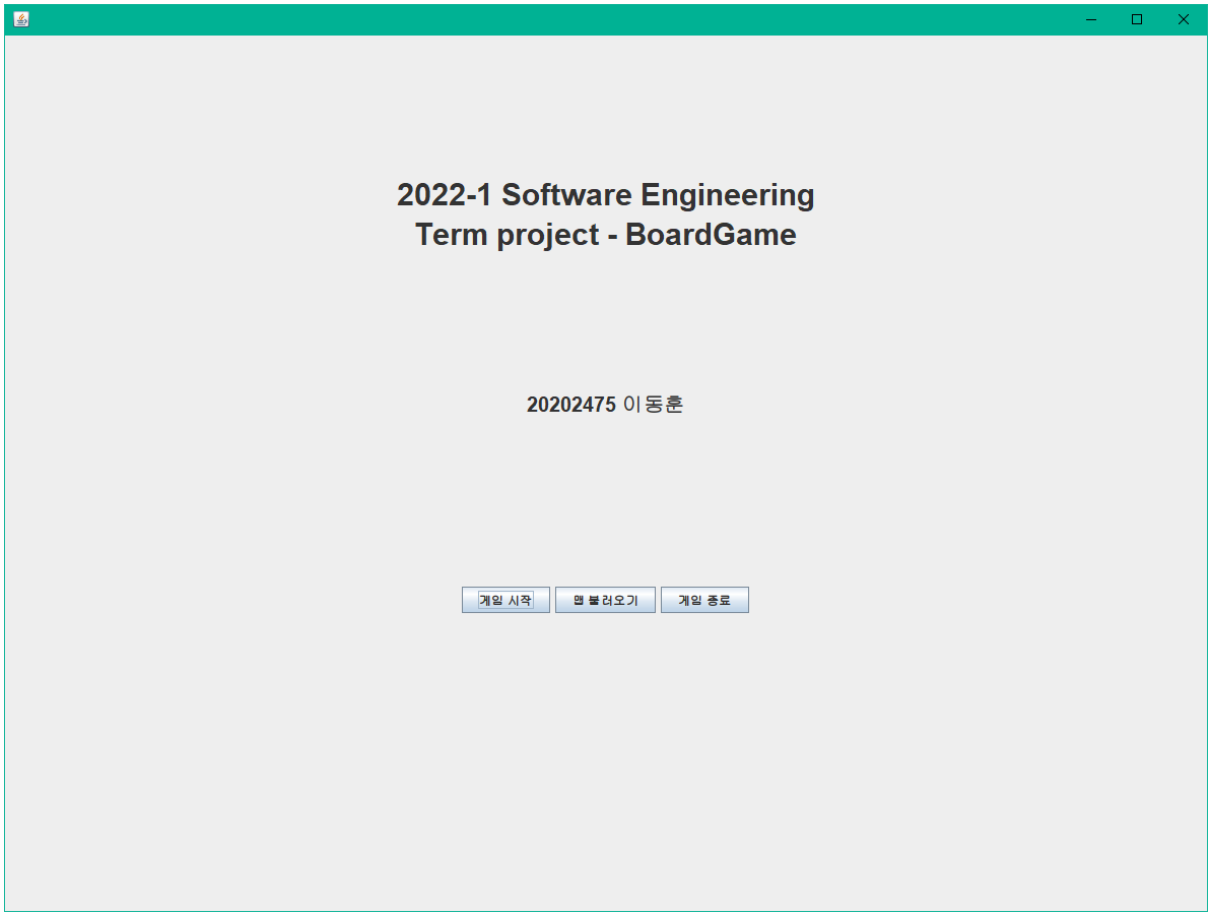
	종료함.
	이외의 셀인 경우, 움직일 수 있는 방향인지 확인 후 가능하다면 움직이고, 그렇지 않다면 횟수를 다시 늘려 움직이지 않음.
 public BoardGame moveAfter() { Cell currentCell = game.getBoard().getCells().getCell(game.getCurrentPlayer().getPosition()); if (currentCell.isEnd()) { game.getCurrentPlayer().setGoalIn(); game.getCurrentPlayer().getStatus().setEndBonus(game.getRank()); game.addRank(); game.setGoalInFlag(); } else if (currentCell.isPdriver()) { game.getCurrentPlayer().getStatus().addPdriver(); } else if (currentCell.isHammer()) { game.getCurrentPlayer().getStatus().addHammer(); } else if (currentCell.isSaw()) { game.getCurrentPlayer().getStatus().addSaw(); } return game; }	움직이고 난 이후 도착한 셀이 도구나 도착 셀일 경우 해당 셀에 맞는 처리
public BoardGame startTurn() { game.startTurn(); return game; } public BoardGame endTurn() { game.endTurn(); return game; }	차례의 시작과 끝을 설정하는 메소드
private static void backToStart(gameView gameV) { gameV.onExit(); programExecute(); } private static void gameEnter(startView startV) { startV.onExit(); gameV = new gameView(); gameV.exitButton.addActionListener(e -> backToStart(gameV)); }	backToStart - gameView에서 돌아가기를 클릭하면 호출되는 메소드, startView가 다시 실행됨 gameEnter – gameView 화면으로 이동하는 메소드

	newMapLoad – 맵 불러오기 버튼을 누르면 실행되는 메소드, 새 맵을 저장함 gameFinish – 게임이 종료되고 난 후 resultView를 호출하기 위한 메소드 버튼이 클릭되면 startView로 돌아감
	gameExit – 종료하기 버튼을 누르면 실행되는 메소드, 프로그램이 종료됨 programExecute – 메인화면인 startView가 실행되는 메소드 버튼에 리스너를 달아서 해당 버튼에 맞는 동작을 정의함
	Main method – programExecute 메소드 호출

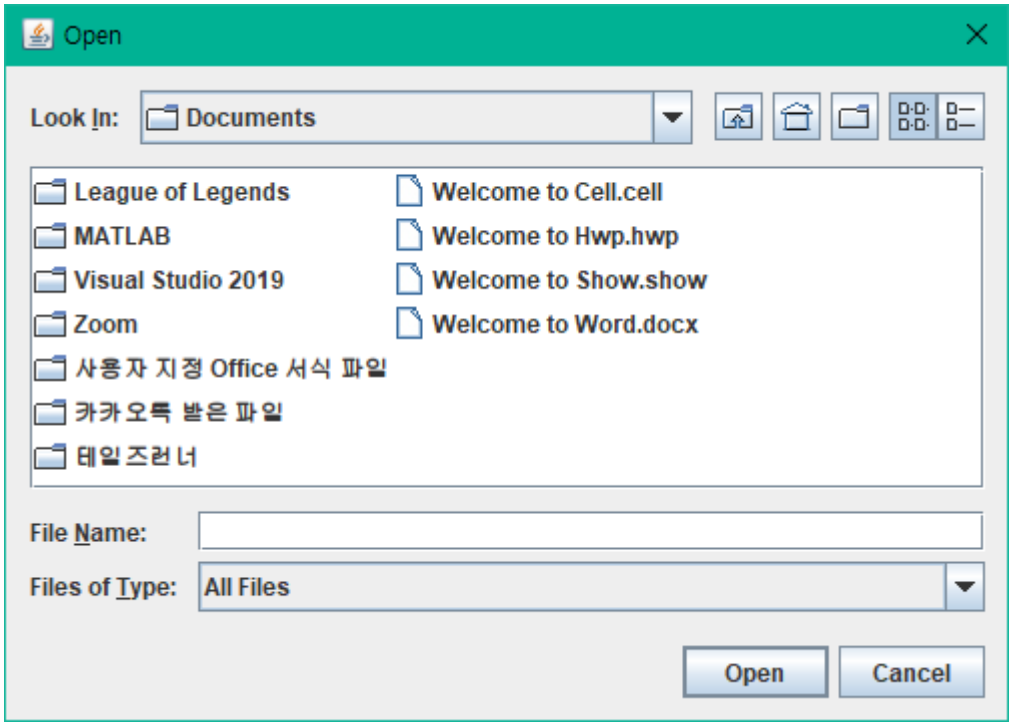
2) 프로그램 사용 방법

1. 명령 프롬프트를 열고 패키지 바로 바깥 폴더에 들어간다.
2. javac -encoding utf-8 boardGame/*.java 를 입력하여 컴파일한다.
3. java boardGame/main 을 입력해 실행한다.
4. 게임을 진행한다.

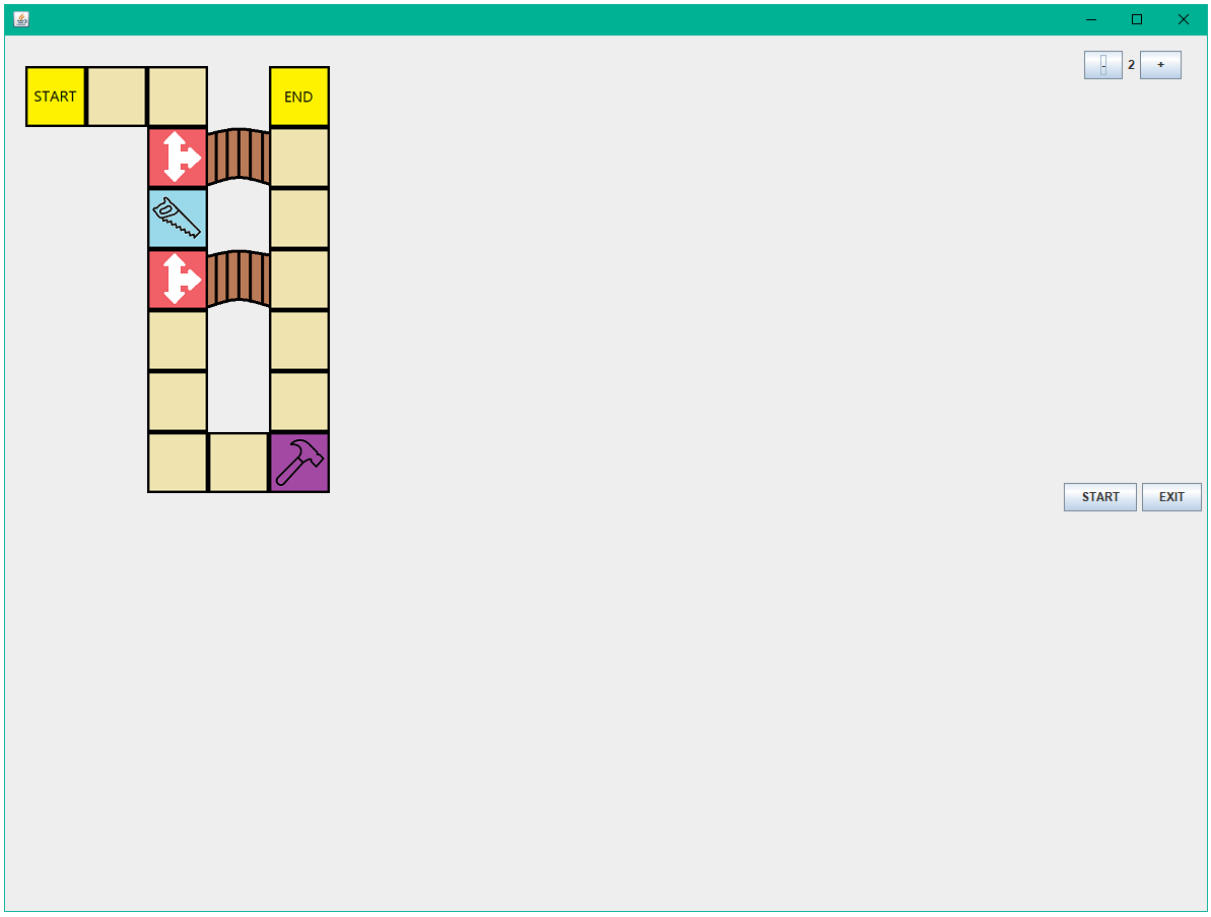
3) 테스트 결과 화면



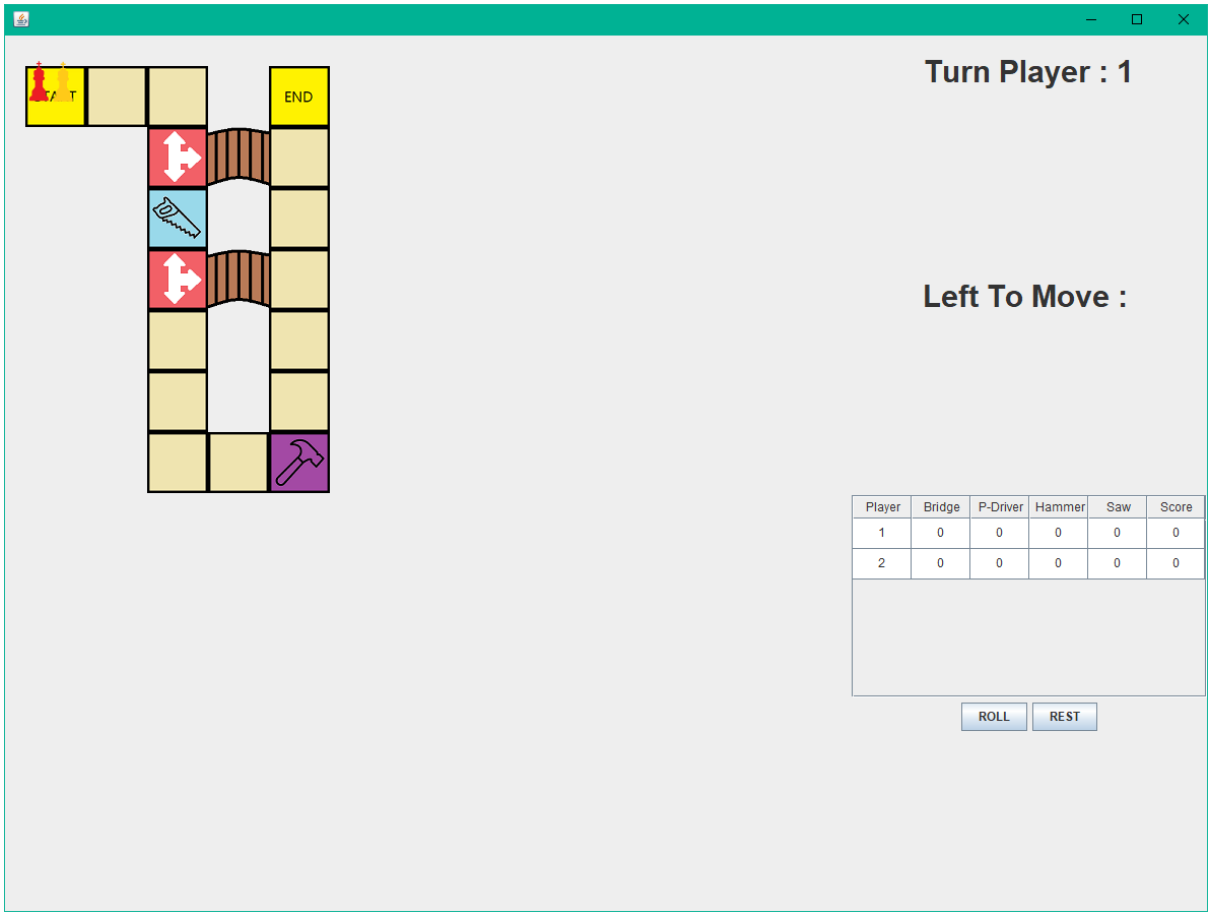
프로그램 시작 시 나오는 메인 화면



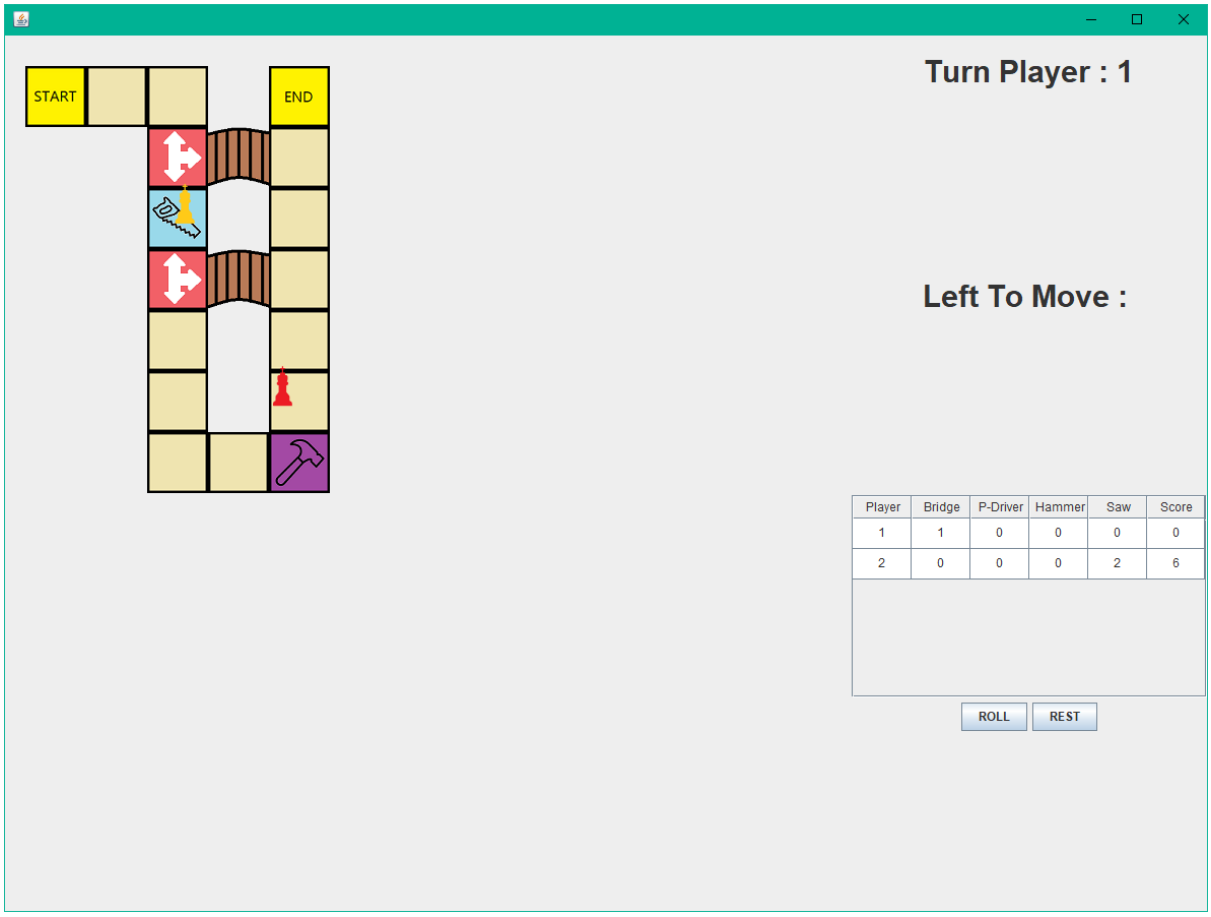
맵 불러오기 버튼 클릭 시 나오는 file dialog 창



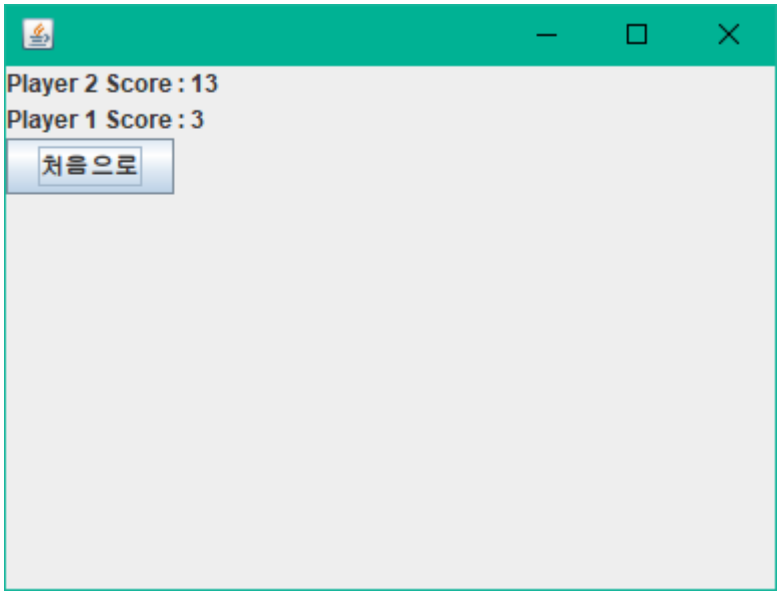
게임 시작 버튼 클릭 시 화면


END
Turn Player : 1
Left To Move :
Player
Bridge
P-Driver
Hammer
Saw
Score
1
0
0
0
0
0
2
0
0
0
0
0
ROLL
REST

스타트 버튼 클릭 시 화면



게임 진행 중 화면



게임 종료 후 화면